THE UNIVERSITY of ADELAIDE

Towards Quality-centric Design and Evaluation of Big Data Cyber Security Analytics Systems

Author: **Faheem Ullah**

Principle Supervisor: Prof. Dr. Muhammad Ali Babar

Co-supervisor: Dr. Mingyu Guo

A thesis submitted for the degree of Doctoral of Philosophy

in

Centre for Research on Engineering Software Technologies (CREST)

School of Computer Science

Faculty of Engineering, Computer and Mathematical Sciences

The University of Adelaide

February 2020

Contents

List of Figures

List of Tables

Abstract

Towards Quality-centric Design and Evaluation of Big Data Cyber Security Analytics Systems

by Faheem Ullah

Big Data Cyber Security Analytics (BDCA) systems are a new breed of software systems that leverage big data technologies to collect, store, and analyse a large volume of security events data for detecting cyber-attacks. To detect sophisticated and complex cyber-attacks, many organizations are rapidly adopting BDCA systems to analyse a large volume of security events data in diverse formats from multiple sources such as network devices, software applications and honeypots. BDCA systems are a complex class of software systems that are expected to fulfil a certain class of quality attributes such as *response time*, *accuracy*, and *scalability*. Given the increasing volume, velocity, and heterogeneity of security events data, BDCA systems present unique design challenges and new research and development opportunities for providing suitable design and evaluation support.

However, most of the research efforts have focused on algorithmic solutions (e.g., data filtering and feature selection) for optimizing response time, accuracy, and scalability of BDCA systems. Hence, there is an important need of research efforts aimed at providing suitable design knowledge (e.g., architectural tactics and design guidelines) for BDCA systems. More research efforts also need to be invested in exploiting the optimization opportunities (e.g., selection of components) offered by the architectural design of BDCA systems for optimizing response time, accuracy, and scalability. This thesis aims at contributing to the growing body of design and evaluation knowledge for BDCA systems by gathering/devising, implementing, and evaluating a set of quality-centric design approaches for optimizing response time, accuracy, and scalability of BDCA systems. This thesis advances the domain of BDCA systems' design and evaluation knowledge by making the following contributions.

- Design, conduct, and report a systematic literature review of the state-of-the-art BDCA systems to identify the most important quality attributes and codify architectural tactics for BDCA systems
- Quantify the impact of architectural tactics on the *accuracy* and *response time* of a BDCA system through a systematically designed experimentation, which leads to the formulation of tactics-specific guidelines for designing BDCA systems
- Present and evaluate a design approach for determining an architecture for a BDCA system at design time that offers optimal *accuracy* and *response time*
- Present and evaluate an architecture-driven adaptation approaches for (re)composing a BDCA system at runtime with a set of components to ensure optimal *accuracy* and *response time* in the face of changes in the operating environment of the system
- Present and evaluate a scalable fuzzy rule based approach to correlate security event data with the components of a BDCA system for (re)composing a BDCA system at runtime to

ensure optimal *accuracy* and *response time* in the face of changes in the operating environment of the system.

- Investigate the scalability of a BDCA system with the default and modified configuration setting of the underlying big data framework (i.e., Apache Spark) to explore and subsequently exploit configuration setting for optimizing *scalability*

Declaration

I, **Faheem Ullah**, certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I acknowledge that copyright of published works contained within this thesis resides with the copyright holder(s) of those works.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Date: 14/02/2020

Acknowledgement

I would like to express my greatest gratitude to my principal supervisor, Prof. Ali Babar, for giving me an opportunity to carry out PhD research under his supervision. Prof. Ali has been extremely kind and supportive throughout this amazing journey of PhD. He has taught me the very basics of research and helped me through thick and thin to walk the path of PhD. His support and contributions in the form of brainstorming research problems, comments on papers, helping me engage with industry and other researchers, involving me in student projects, and motivating me to do high-quality research has been extremely valuable.

I also thank my co-supervisor, Dr. Mingyu Guo, for his insightful comments on different pieces of the research carried out during my PhD candidature.

I am grateful to all members of the CREST team for their valuable research discussions, useful comments on my research papers, and above all being good friends. A special thanks goes to Mojtaba Shahin, Treit Le, and Chadni Islam for their valuable feedback and discussions related to my research papers.

Finally, I would like to thank my family for their encouragement and support.

Chapter 1

Introduction

The volume, velocity, and variety of digital data are increasing. It is expected that by 2020, the amount of digital data will reach 40 trillion gigabytes, which was merely 1.2 trillion gigabytes in 2010 [7]. Each second, around 1.7 MB data is generated per person [8]. Considering the famous quote "data is the new oil of the digital economy", the proportion of data considered for analysis has jumped from 0.5% in 2012 to 37% in 2019 [7, 9]. With the increasing volume, velocity, and variety of data, the traditional software systems (e.g., relational database and data warehouse) are rapidly becoming unable to collect, store, and analyse the large volume of data. Therefore, organizations are pivoting towards leveraging big data technologies (e.g., Hadoop, Spark, and Cassandra) to deal with the massive volume, velocity, and variety of data referred to as 'big data' [10]. It is reported that 97.2% of organizations are investing in big data [11]. Although different organizations (e.g., Google, Microsoft, and IBM) define 'big data' in their own ways, there are at least three factors that usually characterise big data – the volume of data, the variety of data, and the technologies used to analyse the data [12].

Similar to other domains such as bioinformatics [13] and healthcare [14], the role of big data technologies in cyber security domain is on the rise. The significance of big data technologies in the field of cyber security was first highlighted in a Cloud Security Alliance in 2013 [15]. The report [15] published by Cloud Security Alliance emphasized the dire need of supplementing the traditional cyber security systems (e.g., intrusion detection system and malware detection system) with big data technologies to enable these systems to deal with the massive volume of security event data (e.g., NetFlow, firewall logs, and packet data). Even in 2013, an organization as large as Hewlett-Packard (HP) generated around one trillion security events per day [15], while merely a 1 Gbps continuous network traffic can make a traditional intrusion detection system obsolete [16]. The increasing volume of security event data and limitations of traditional cyber security systems led to the merger of big data technologies and cyber security systems, which has led the creation of a new breed of software systems called **B**ig **D**ata **C**yber Security **A**nalytics (BDCA) system. A BDCA system is defined as "*A system that leverages big data technologies for collecting, storing, and analysing a large volume of security event data to protect organizational networks, computers, and data from unauthorized access, damage, or attack*" [1].

According to Cisco, the global IP traffic will increase from 1.5 Zettabyte in 2017 to 4.8 Zettabyte in 2022 [17]. Similarly, Cisco estimates that the number of devices connected to the internet will increase from 18 billion in 2017 to 28.5 billion in 2022 [17]. Both IP traffic and logs generated by the devices are fruitful sources for security analytics [17]. Hence, organizations are enhancing their data management and data processing capabilities to collect and analyse the large volume of data for extracting security insight to protect their cyber assets. In other words, organizations are increasingly realizing the importance of BDCA. This is evident from a large-scale study conducted with 330 organizations from over 50 countries, which reveals that around 88% of organizations consider BDCA as very important for cyber protection [18]. Furthermore, the study reveals that organizations employing BDCA systems report 53% high benefits (as compared to traditional cyber security solutions) in terms of detecting and

responding to cyber-attacks. A more recent study conducted with 479 security executives concludes that 69% of organizations employing BDCA report significant improvement in their cyber knowledge management capabilities [19]. The study also shows that 72% of organizations that deployed BDCA systems observed significant improvement in their ability to respond quickly and accurately to cyber-attacks. To exemplify the gain with BDCA, Zions Bancorporation conducted a study, which revealed that their traditional Security Information and Event Management System (SIEM) system takes around 20-60 minutes to query a month of security event data while the same task takes about one minute when using Hadoop technology [20].

BDCA systems possess certain characteristics that distinguish them from traditional cyber security systems. These characteristics include: (1) *Monitoring diverse assets:* BDCA systems monitor diverse set of activities on an organization's Information and Communication Technology (ICT) assets such as computing machines, data storage systems, end-user applications, and computing networks [21] (2) *Security event data integration*: BDCA systems often integrate and correlate security event data from multiple security systems (e.g., IDS, anti-virus and firewall) to gain a holistic understanding of the cyberspace surrounding organizational ICT assets [15, 22] (3) *Handling large volume of security event data*: The exponential growth in cyber activities over the years has made it challenging for organization to store and analyse large amounts of security event data. Hence, BDCA system leverages big data storage and processing technologies (e.g., Cassandra and Hadoop) to handle the storage and processing of such large volume of data [21] (4) *Enabling real-time and deep analytics*: The traditional tools (e.g., a relational database) often become a bottleneck in performing real-time and holistic security analytics. A BDCA system overcomes the bottleneck by enabling near real-time and deep analytics of security event data to reveal hidden patterns and detect low and slow attacks such as Advanced Persistent Threats (APT) [23] (5) *Analysing heterogenous data:* The security event data collected from various sources is often in different formats. In addition to structured data, a BDCA system has the capability to analyse large volumes of semi-structured and unstructured data (e.g., text in logs files) to extract valuable security insight from the data [24].

BDCA systems are divided into two main categories based on their attack detection capability – Generic BDCA systems and Specific BDCA systems [1]. *Generic BDCA systems* aim to detect a variety of attacks such as SQL injection, Cross Site Scripting, and brute force. Examples of Generic BDCA systems include intrusion detection system (e.g., [25, 26]) and alert correlation system (e.g., [27, 28]) designed using big data technologies. *Specific BDCA systems* are focussed on detecting a specific type of attack such as phishing and botnet. Examples of Specific BDCA systems include phishing detection system (e.g., [29, 30]) and botnet detection system (e.g., [31, 32]) designed using big data technologies.

Figure 1.1 shows an overview of a typical BDCA system. The data collection tools (e.g., Wireshark) collect security event data from different sources such as network routers and honeypots. The collected data is then prepared by removing redundancy, missing values, incorrect values, and so on. Furthermore, the features considered most important for attack detection are selected and extracted from the data. The prepared data is analysed using anomaly-based and/or signature-based attack detection techniques to generate alerts. The alerts are then processed to remove false alarms and correlate the alerts to generate alerts related to more complex attacks. The final alerts are used by security operators to respond to the attacks.

Like any software system, certain quality attributes are expected in a BDCA system. A quality attribute of a software system is defined as "a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders." [33]. It has been reported that *response time*,

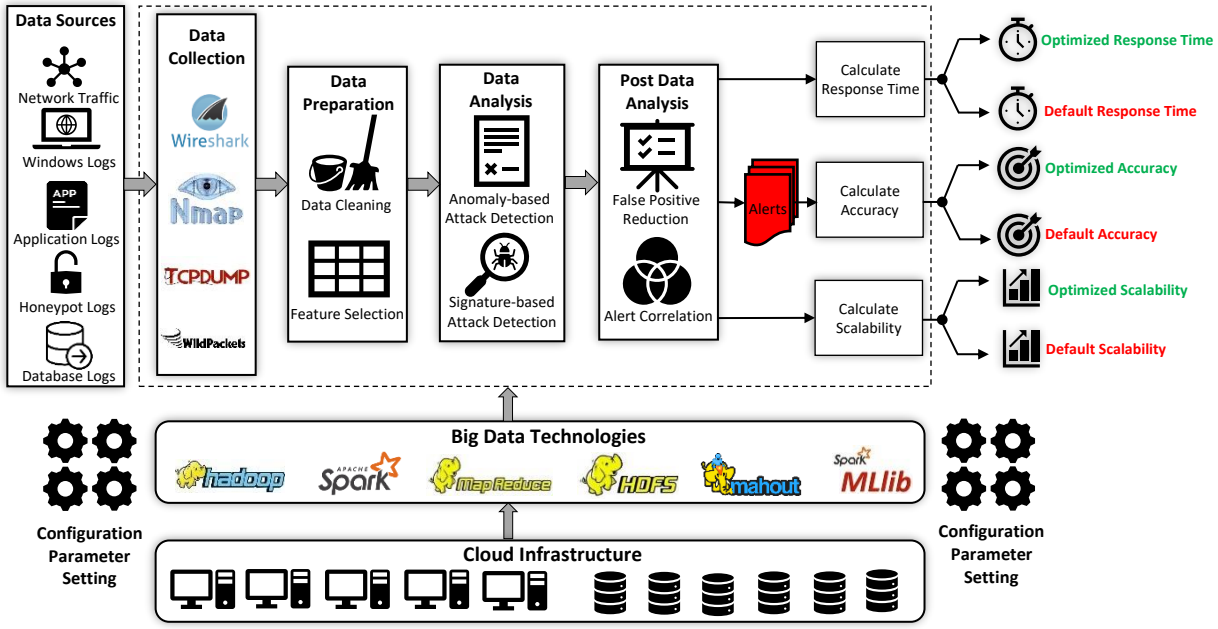


Figure 1. 1 An overview of a typical BDCA system

accuracy, and *scalability* are the three most important quality attributes of a BDCA system [1]. *Response time* measures how quickly a BDCA system prepares and analyses data to generate alerts. A near real-time response is important because the longer an attack goes undetected, the greater is the damage caused [19, 22]. For example, an international consulting company, Wipro, fell victim to a phishing attack in 2019 [34]. The attackers aimed to exfiltrate customer's data. However, Wipro's cyber security systems and security operators swiftly detected and contained the attack before it could exfiltrate sensitive data. *Accuracy* assesses how accurately a BDCA system generates alerts. In other words, accuracy measures whether there is any malicious activity not flagged as an alert and any benign activity flagged as an alert by a BDCA system. A BDCA system should be accurate in detecting malicious activities. This is because the very aim of a BDCA system is that a malicious activity should not go undetected through a BDCA system. At the same time, a BDCA system should not generate floods of false alarms, which consumes the precious time of security operators for attending to meaningless alerts [21]. *Scalability* measures how easily a BDCA system can increase/decrease its processing capacity to cope with the changes in the volume and velocity of security event data. The volume of security event data is rapidly increasing, which requires a BDCA system to scale up (e.g., by adding more computational power) to process data without any impact on the response time [35, 36].

1.1 Research Objectives and Questions

Big data technologies offer support for achieving low response time, high accuracy, and high scalability in BDCA. As exemplified earlier, the use of Apache Hadoop reduces the response time from 20-60 min to one minute [20]. The use of Apache Spark reduces the response time of a BDCA system analysing 268 GB security event data from 180 min to 14 min [29]. Similarly, Apache Yarn is used to achieve high scalability [37]. In addition to industry, academic research has focussed on proposing approaches for optimizing response time, accuracy, and scalability of BDCA systems. Existing studies primarily propose machine learning-based approaches [26, 38-41], data preparation techniques [42-45], feature selection methods [31, 40], and cloud-based solutions [31, 46] for optimizing the three quality attributes of a BDCA system. For instance, [42] and [43] present approaches to limit the collection of security event data to only first 15 KB and 100 sec of a network connection and execution process respectively

to reduce data size, which leads to lower data processing time. Some studies (e.g., [39-41]) compare the accuracy of machine learning algorithms used for anomaly-based attack detection. Similarly, [28] and [46] present cloud-based solutions that monitor CPU utilization to balance the workload among computing nodes and burst the newly arrived workload to a new cluster.

It has been proclaimed that the software architecture of BDCA systems needs to be explored to identify and exploit architectural opportunities for optimizing response time, accuracy, and scalability [21]. Software architectural design underlines the components comprising a BDCA system and the relationship among the components [33]. Several studies [33, 47, 48] highlight the role of software architectural design in achieving various quality attributes. However, given that BDCA systems are a new and complex breed of software systems, there is a lack of systematic design knowledge, such as quality requirements and architectural tactics, for designing BDCA systems. Suck lack of design knowledge makes it challenging to understand and subsequently exploit the architectural level opportunities (e.g., selection of architectural components) for optimizing response time, accuracy, and scalability. The goal of this thesis is to fill this gap by proposing, implementing, and evaluating design knowledge and a set of quality-centric approaches for designing BDCA systems that exhibit an optimal level of response time, accuracy, and scalability.

> ***Problem Statement:*** *Response time, accuracy, and scalability are the three most important quality attributes of a Big Data Cyber Security Analytics (BDCA) system. Given that designing BDCA systems is a challenging task, there is a lack of focus on exploring the design space of BDCA systems and exploiting the opportunities offered by the architectural design for optimizing response time, accuracy, and scalability. Therefore, it is important to identify, investigate, and exploit the architectural design level opportunities of a BDCA system to optimize response time, accuracy, and scalability.*

To realize the goals of the thesis, we address the following research questions.

RQ1: **What are the most important quality attributes and the architectural tactics for BDCA systems?**

Since BDCA is a relatively new breed of software systems, there is a lack of consensus on what are the most important quality attributes of a BDCA system. Furthermore, there is a paucity of systematic efforts aimed at identifying and codifying architectural tactics used in the state-of-the-art for achieving various quality attributes in a BDCA system. Therefore, it is important to systematically review the state-of-the-art BDCA systems to (i) identify the most important quality attributes and the rationale behind their significance and (ii) identify and codify the architectural tactics and associated opportunities for achieving and optimizing various quality attributes such as response time, accuracy, and scalability.

RQ2: **What is the impact of various architectural tactics on the accuracy and response time of a BDCA system?**

It has been observed that various architectural tactics are randomly used without considering the underlying contextual factors such as quality of security event data, execution mode of a BDCA system (e.g., pseudo-distributed or fully-distributed), and machine learning models employed in BDCA systems. Hence, there is a need for an empirical investigation to quantify the impact of certain architectural tactics on the accuracy and response time with respect to the contextual factors. The findings from an empirical investigation is expected to provide guidelines for optimally using the architectural tactics that can impact a set of desired quality attributes.

RQ3: **How to design a BDCA system that best satisfy quality goals, design dependencies, and design constraints?**

The architectural tactics used for designing a BDCA system depend on other tactics and are subject to various design constraints (e.g., availability of computational power). Moreover, the effectiveness of the tactics depends on various contextual factors and have implications for multiple quality attributes (e.g., response time, reliability, and interoperability). Such design complexities make designing a BDCA system a challenging undertaking. Hence, there is a need for a design approach that helps architects to understand such design complexities and design a BDCA system that best satisfies quality goals, design dependencies, and design constraints.

RQ4: **How to enable a BDCA system to ensure optimal accuracy and response time in the face of changes in the operating environment?**

The operating environment (e.g., an enterprise network) where a BDCA is deployed comes across various types of changes such as a change in the quality and velocity of security event data. Such changes may degrade a BDCA system's quality attributes, e.g., accuracy and response time, which may be unequally impacted by the changes in the operating environment. For instance, an increase in the velocity of security event data may result in a negative impact on response time but keeping accuracy largely intact. Thus, there is a need for adaptation approaches that can adapt a BDCA system according to the changes in its operating environment to ensure an optimal level of accuracy and response time.

RQ5: **What is the impact of tuning configuration parameters of a big data framework on the scalability of a BDCA system and how to select a configuration that optimizes scalability?**

A big data analytical framework (e.g., Apache Spark) is the most integral part of any BDCA system. By integral part, we mean a part without which a BDCA system cannot function. Since a BDCA system requires a big data analytical framework for distributing the storage and processing of the data across computing nodes, big data analytics framework is termed as the most integral part of the BDCA system. Big data analytical frameworks offer several configuration parameters, which impact how a framework process data. Users can tune a framework's configuration parameters for specific use cases (e.g., optimal scalability) to get the maximum benefit from a framework's abilities. Given that each framework has several configuration parameters (e.g., 187 parameters in Apache Spark), it is challenging to manually tune the parameters. Therefore, it is essential to first investigate the impact of configuration parameters on the scalability of a BDCA system and then develop a tuning approach for selecting a configuration that ensures optimal scalability.

1.2 Research Methodology

Given this thesis reports the research that has explored and addressed different aspects of architectural design and big data technological considerations for BDCA systems, each chapter of this thesis has its own flavour while contributing to the overall goal of the thesis. Hence, the details of the methodological aspects of different pieces of the research for this thesis are reported in the relevant chapters. Here, we provide an overview of the research methodology employed to answer the research questions presented in the previous section.

This thesis endeavours to propose, implement, and evaluate design knowledge and quality-centric approaches for optimizing response time, accuracy, and scalability of a BDCA system. Chapter 2, being the only theoretical chapter, follows the well-known Systematic Literature Review (SLR) method [49] to answer the research question. Except Chapter 2, all chapters of this thesis either experimentally

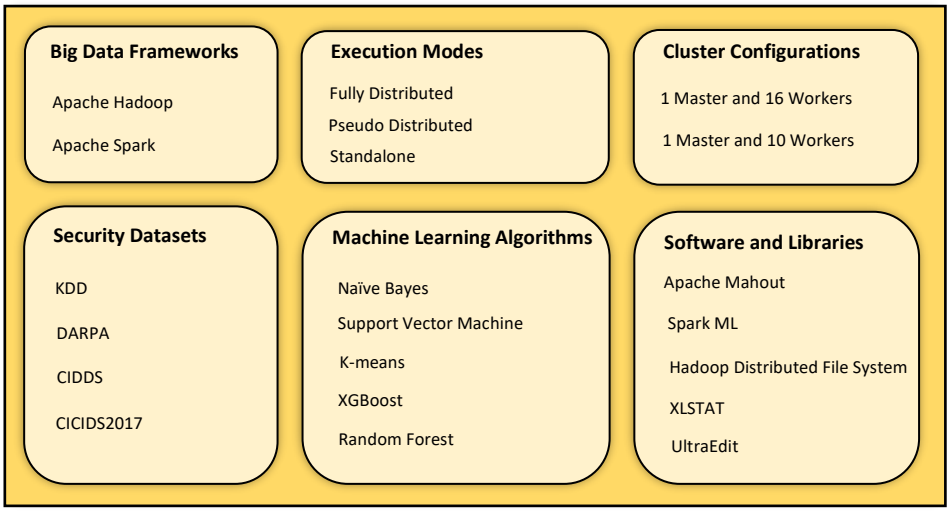


Figure 1. 2 Technologies, datasets, and setups used in the research reported in this thesis

investigate a phenomenon (e.g., impact of architectural tactics in Chapter 3) or propose and implement an approach, and then evaluate the approach based on the experimental results. For implementing the approaches and investigating a certain phenomenon, we used several technologies, datasets, and experimental setups shown in Figure 1.2. We mainly used two big data frameworks (i.e., Apache Hadoop [50] and Apache Spark [51]) for the implementation of the BDCA system. These two frameworks are the most widely used big data frameworks both in the domain of BDCA (as evident from [1]) and big data analytics in general [10]. A machine learning algorithm is employed in the BDCA system for anomaly-based attack detection. Therefore, we implemented and evaluated our approaches with five machine learning algorithms – Naïve Bayes, Support Vector Machine, Random Forest, XGBoost, and K-means. Since there exists a multitude of machine learning algorithms, we selected the five algorithms based on the following criteria: (i) an algorithm is already successfully applied in BDCA systems [1] (ii) an algorithm is widely used in Kaggle competitions[1], which indicates a widespread use and efficiency of the algorithm in domains other than BDCA too and (iii) an algorithm is available in machine learning libraries (i.e., Mahout and SparkML) for Hadoop and Spark. We used Apache Mahout [52] and SparkML [53] libraries for the distributed implementation of the machine learning algorithms. For data storage, we used Hadoop Distributed File System (HDFS) [54]. For manual navigation through the security datasets, we used Ultraedit[2]. We used XLSTAT[3] for statistical analysis of the experimental results. A BDCA system can be executed in three different execution modes – standalone, pseudo-distributed, and fully-distributed mode (details available in Chapter 3). All experiments are conducted in a fully distributed mode, which is the most reliable and real-world execution mode [50]. In addition to using the fully distributed mode, some experiments (e.g., in Chapter 3 and Chapter 6) are also conducted in standalone and pseudo distributed mode as per the requirement of the studies reported in the different chapters of this thesis. For the fully distributed mode, we ran the experiments on two types of cluster configured on an OpenStack cloud. The first type of cluster consisted of one master and 16 worker nodes; the second type consisted of one master and 10 worker nodes. For all the reported experiments, we used four widely used security event datasets - KDD [55], DARPA [56], CIDDS [57], and CICIDS2017 [58]). We selected these four datasets based on the following criteria: (i) the datasets should vary from each other in terms of quality and quantity of

[1] https://www.kaggle.com/competitions
[2] https://www.ultraedit.com/
[3] https://www.xlstat.com/en/

data in each dataset (ii) the datasets vary in terms of attacks in the dataset (iii) the publication dates of the datasets are diverse (i.e., 1998-2017) and (iv) the datasets are widely used for evaluating BDCA systems [59]. Such diversity in datasets adds to the rigour and generalization of our findings. Similar to other research in BDCA, the findings reported in this thesis face limitations and/or threats to validity. Given that the limitations are relevant to the study reported in each chapter, we present the limitations in the relevant chapters.

1.3 Thesis Contributions

This thesis makes the following six contributions.

1. A systematic literature review of the state-of-the-art BDCA systems (Chapter 2)

    - Identification of the 12 most important quality attributes of a BDCA system and the motivation behind their importance for a BDCA system
    - Identification and codification of 17 architectural tactics for achieving various quality attributes in a BDCA system

2. Quantification of the impact of architectural tactics on the accuracy and response time of a BDCA system (Chapter 3)

    - Design of a systematic experimentation approach for quantifying the impact of architectural tactics
    - Empirical quantification of the impact of architectural tactics on the accuracy and response time of a BDCA system that leads to the formulation of evidence-based guidelines for designing BDCA systems

3. A tactics-driven approach for designing BDCA systems (Chapter 4)

    - Design guidance models to reveal quality implications, design constraints, and design dependencies for architectural tactics used in the design of a BDCA system
    - A heuristic-based approach that uses the guidance models to determine an optimal architecture for a BDCA system
    - Evaluation of the approach in a distributed setting through rigorous experimentation

4. An architecture-driven adaptation approach (*ADABTics*) for BDCA systems (Chapter 5)

    - Demonstration of the impact of change in the operating environment on the accuracy and response time of a BDCA system
    - A component-based architecture for a BDCA system
    - An architecture-driven adaptation approach that (re)composes a BDCA system at runtime with a set of components to ensure optimal accuracy and response time
    - Evaluation of the adaptation approach in standalone, pseudo-distributed, and fully distributed mode using four security datasets

5. A scalable adaptation approach (*QuickAdapt*) for BDCA systems (Chapter 6)

- Formulation of fuzzy rules for runtime selection of components of a BDCA system
- A scalable adaptation approach that uses the fuzzy rules to (re)compose a BDCA system at runtime with a set of components to ensure optimal accuracy and response time
- Evaluation and demonstration of the effectiveness of the adaptation approach through comprehensive experimentation

6. Investigation and optimization of the scalability of BDCA systems (Chapter 7)

- An empirical investigation of how a BDCA system scales with the default configuration setting of the underlying big data framework
- Identification of the configuration parameters of the big data framework that significantly impacts the scalability of a BDCA system
- A parameter tuning approach (*SCALER*) that tunes the configuration parameters of a big data framework to ensure optimal scalability for a BDCA system running on the top of the big data framework

1.4 Publications and Thesis Organization

The core chapters of this thesis are derived from research papers already published (or under review) during my PhD candidature in collaboration with my supervisor. However, I have carried out the main research, albeit, this thesis uses a standard pronoun of "We" for reporting collaborative research in computer science/software engineering. Figure 1.3 shows an overview of the thesis as described below. The dashed boxes in Figure 1.3 specify subparts that are focussed in the respective chapter. For example, Chapter 1 is focussed on two parts – quality attributes and architectural tactics for BDCA systems.

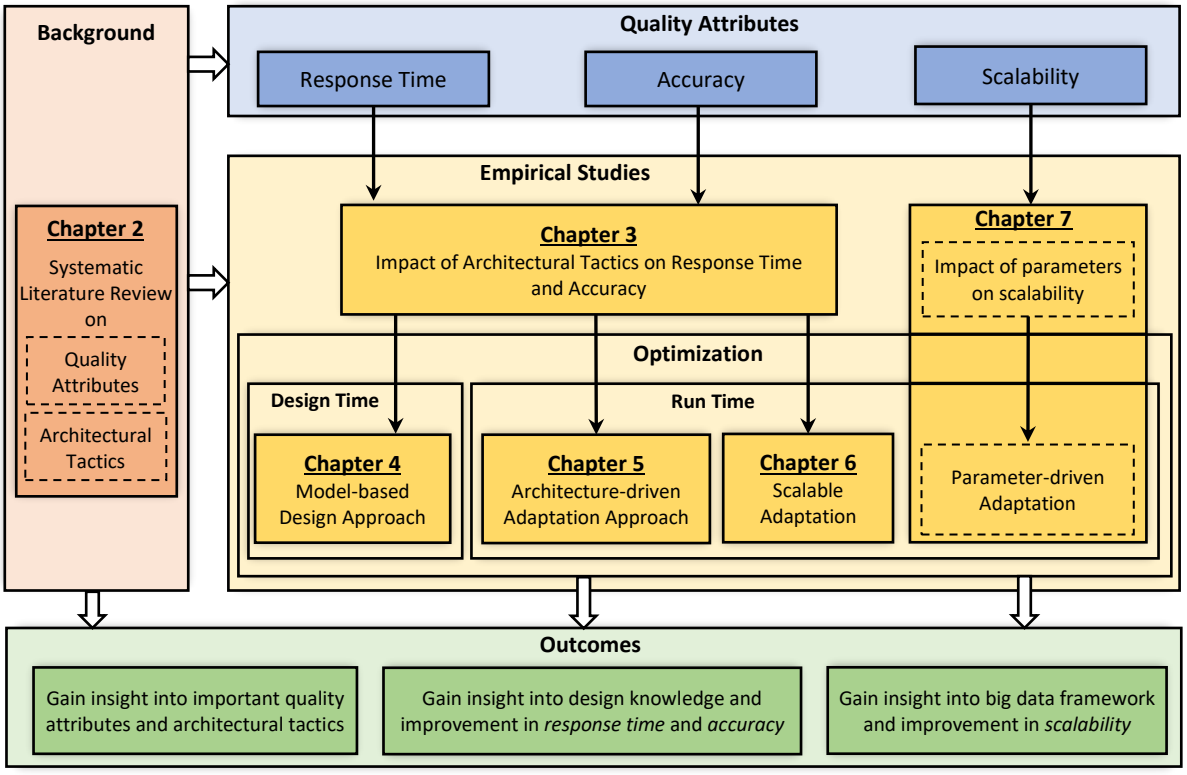


Figure 1. 3 An overview of the thesis

Chapter 1 – This chapter describes the background of BDCA systems, the motivation behind the thesis, the problem statement, the research questions, the research methodology, the contribution of the thesis, and the organization and associated publications of the thesis.

Chapter 2 – This chapter addresses research question ***RQ1*** by reporting results of a systematic literature review aimed at identifying the quality attributes, identifying and codifying architectural tactics, and unfolding areas for future research in BDCA systems including optimization opportunities. The chapter has been previously published as:

> ❶ **Faheem Ullah** and Muhammad Ali Babar, *Architectural Tactics for Big Data Cybersecurity Analytics: A Review*, Journal of Systems and Software (JSS), Volume 151, Pages: 81-118, 2019, Elsevier. [Impact Factor (2018): **2.55**, Core ranking: **rank A**]

Chapter 3 – This chapter answers the research question ***RQ2***. The chapter first reports the design of a systematic experimentation framework used for quantifying the impact of architectural tactics. The chapter then presents the results of a large-scale experiment aimed at quantifying the impact of architectural tactics on the accuracy and response time of a BDCA system. Based on the quantified impact, the chapter also presents design guidelines for the optimal use of the studied tactics. The chapter has already been published as:

> ❷ **Faheem Ullah** and Muhammad Ali Babar, *Quantifying the Impact of Design Strategies for Big Data Cyber Security Analytics Systems: An Empirical Investigation*, In Proceedings of the 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), Gold Coast, Australia, 2019. [Core ranking: **rank B**, Acceptance rate: **30%** (60/200)]

Chapter 4 – This chapter address ***RQ3*** by presenting the design approach - *TACTics*. The chapter first presents guidance models (first part of *TACTics*) for understanding various design areas, which is followed by a heuristic-based design method (second part of *TACTics*) to determine an optimal design for a BDCA system. The chapter is based on the following paper:

> ❸ **Faheem Ullah** and Muhammad Ali Babar, *A Tactics-driven Approach for Designing Big Data Cyber Security Analytics Systems*, (under review): Journal of Systems and Software (JSS), 2019, Elsevier. [Impact Factor (2018): **2.55**, Core ranking: **rank A**]

Chapter 5 – This chapter presents *ADABTics* – an adaptation approach for ensuring optimal accuracy and response time in the face of changes in the operating environment to answer ***RQ4***. The chapter first reports a component-based architecture for BDCA, which is followed by the adaptation approach that exploits the component-based architecture. This chapter has previously appeared in:

> ❹ **Faheem Ullah** and Muhammad Ali Babar, *An Architecture-driven Adaptation Approach for Big Data Cyber Security Analytics*, In Proceedings of the International Conference on Software Architecture (ICSA), Hamburg, Germany, 2019. [Core ranking: **new**, Acceptance rate: **21.8%** (21/96)]

Chapter 6 – This chapter address ***RQ4*** by presenting an adaptation approach (*QuickAdapt*) that relies on fuzzy rules to draw a mapping between the security event data and components of a BDCA system. Hence, reducing the adaptation time by a significant margin. The chapter has been already published as:

> ❺ **Faheem Ullah** and Muhammad Ali Babar, *QuickAdapt: Scalable Adaptation for Big Daa Cyber Security Analytics*, In Proceedings of the 24th International Conference on Engineering of Complex Computer Systems (ICECCS), Guangzhou, China, 2019. [Core ranking: **A**, Acceptance rate: **31.8%** (28/88)]

Chapter 7 – This chapter address ***RQ5***. The chapter first reports on how a BDCA system scales with the default configuration setting of a big data framework. The chapter then investigates the impact of various configuration parameters on the scalability. Finally, the chapter presents a parameter tuning approach that automatically selects the configuration setting with the optimal scalability. The chapter is based on the following paper:

> ❻ **Faheem Ullah** and Muhammad Ali Babar, *On the Scalability of Big Data Cyber Security Analytics*, (under review): Journal of Future Generation Computer Systems (FGCS), 2019, Elsevier. [Impact Factor (2018): **5.76**, Core ranking: **rank A**]

Chapter 8 – This chapter concludes the thesis with a summary of the key findings and suggestions for future work.

1.5 Other Publications

In addition to the six publications constituting the core of the thesis, I contributed as first author/co-author of the following four publications (during my PhD candidature), which are not directly used in this thesis.

> ❼ **Faheem Ullah**, Adam Johannes Raft, Mojtaba Shahin, Mansooreh Zahedi, and Muhammad Ali Babar, *Security Support in Continuous Deployment Pipeline*, In Proceedings of 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), Pages: 57-68, Porto, Portugal, 2017. [Core ranking: **rank B**]
>
> ❽ **Faheem Ullah**, Matthew Edwards, Rajiv Ramdhany, Ruzanna Chitchyan, Muhammad Ali Babar, and Awais Rashid, *Data Exfiltration: A Review of External Attack Vectors and Countermeasures*, Journal of Network and Computer Applications (JNCA), Volume 101, Pages: 18-54, 2018, Elsevier. [Impact Factor (2018): **5.27**, Core ranking: **rank A**]
>
> ❾ **Faheem Ullah** and Muhammad Ali Babar, *A Parameter-driven Adaptation Approach for Big Data Cyber Security Analytics*, (under review): 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), South Korea, 2019.
>
> ❿ Bushra Sabir, **Faheem Ullah**, Muhammad Ali Babar, and Raj Gaire, *Machine Learning for Detecting Data Exfiltration: A Review,* (under review): *ACM Computing Surveys,* 2019, ACM Press. [Impact factor (2018): **6.13**, Core ranking: **rank A***]

Chapter 2

A Review of Quality Attributes and Architectural Tactics for Big Data Cyber Security Analytics Systems

Related publication:

This chapter is based on our paper published in *Journal of Systems and Software (JSS 2019)* "Architectural Tactics for Big Data Cybersecurity Analytics: A Review" [1].

Big **D**ata **C**yber Security **A**nalytics (BDCA) systems are a relatively new breed of software systems aimed at leveraging big data technologies (e.g., Hadoop and Spark) for analyzing security events data to protect organizational networks, computers, and data from cyber attacks. Given the growing need and popularity of **B**ig **D**ata **C**yber Security **A**nalytics (BDCA) systems, it is important to systematically discover and document architectural aspects of such systems for building a suitable design space. This chapter aims at identifying the most frequently reported quality attributes and architectural tactics for BDCA systems. We also intend to highlight the research gaps in this area. For realizing this goal, we used the Systematic Literature Review (SLR) method for reviewing 74 papers on BDCA published between 2010 and May 2017. Our review has identified 12 most important quality attributes and codified 17 architectural tactics for BDCA. This chapter also highlights the areas for future research on BDCA and provides some recommendations for practitioners.

2.1 Introduction

Cyber security is aimed at protecting Information and Communication Technology (ICT) infrastructures (i.e., computer hardware, software, networks and data) from unauthorized access and/or disruptions [60]. Cyber security solutions are increasingly leveraging big data technologies for (i) collecting security events data (ii) performing deep security analytics and (iii) providing a consolidated view of security information [15, 22]. A large-scale industrial survey [61] reports that an increasing number of organizations have been realizing the importance and value of big data technologies for protecting their ICT infrastructures against potential cyber attacks. As described in Chapter 1, a cyber security solution that incorporates big data technologies is called **B**ig **D**ata **C**yber Security **A**nalytics (BDCA) system, which is defined as "*A system that leverages big data technologies for collecting, storing, and analysing a large volume of security event data to protect organizational networks, computers, and data from unauthorized access, damage, or attack*".

Given that BDCA systems sit at the intersection of cyber security, big data technologies, and software engineering, it is a challenging undertaking to design and evaluate highly scalable BDCA systems that

can effectively and efficiently manage and analyse a massive amount of security event data [24, 62]. Designers of BDCA systems are expected to consider several unique aspects of BDCA systems such as (i) privacy assurance (ii) authenticity and integrity of security events data (iii) lack of datasets for evaluating security systems (iv) asymmetrical cost of misclassification of security events (v) detecting and correcting adversarial learning and (vi) attack time scale [22, 63]. Like any other large-scale software-intensive system, the achievement of common domain-specific functional and non-functional (i.e., quality attributes) goals should be a key concern while designing BDCA systems. There is a lack of consensus on which are the most critical quality attributes of BDCA systems. Furthermore, there is a paucity of systematic efforts aimed at building guidelines for designing BDCA systems [15, 64].

For such design guidelines, the Software Engineering (SE) community has emphasized the importance of organising design knowledge (i.e., quality attributes, tactics, and patterns) as design spaces [65, 66], which communicates experienced-based design knowledge to software designers. A design space highlights the significance of certain quality attributes (e.g., scalability and reliability) and codifies architectural tactics (i.e., *a reusable design strategy that helps achieve a particular quality attribute* [33]), which can be used for the systematic design of a software system [66]. Recently, there are increasing number of efforts to systematically reviewing and understanding the architectural aspects of software-intensive systems for building and leveraging architectural knowledge (i.e., design spaces) in different domains such as cyber-foraging [67], energy efficiency in cloud [68], ambient assisted living systems [69], and self-adaptive systems [70].

Motivated by such efforts and increasing need and importance of security design knowledge, this chapter reports a systematic study of the state-of-the-art of BDCA systems from an architectural perspective. The chapter aims to identify and codify the most important quality attributes and architectural tactics for BDCA systems. The chapter is expected to communicate experience-based design knowledge to software architects for systematically and efficiently designing BDCA systems. To this end, we have systematically selected and rigorously reviewed 74 relevant papers reporting BDCA systems. We assert that the research reported in this chapter will provide potentially useful knowledge and insights to those researchers and practitioners who are interested in gaining a better understanding of the architectural knowledge (i.e., quality attributes and design choices) used for designing BDCA systems and the identification of the areas where further exploration and experimentation are required.

Contributions: Our analysis and synthesis of the data extracted from the 74 reviewed papers have enabled us to make the following key contributions:

1. Identification of the 12 most important quality attributes for BDCA systems
2. A catalogue of 17 architectural tactics for addressing quality concerns in BDCA systems
3. Identification of potential research areas to advance the state-of-the-art on architecting BDCA systems

Chapter Organization: The rest of this chapter is organized as follows. Section 2.2 presents the research methodology used for conducting this review. Section 2.3 reports meta-analysis of the reviewed papers. Section 2.4 presents the identified critical quality attributes, which is followed by Section 2.5 that presents the codified architectural tactics. Section 2.6 presents the lessons learned, areas for future research, and provides a few recommendations for software architects. Section 2.7 reports the related work. Section 2.8 discusses the limitations of this review. Finally, Section 2.9 concludes the chapter.

2.2 Research Methodology

As discussed in Chapter 1, we used the Systematic Literature Review (SLR) [49] method for conducting this review. An SLR is aimed at systematically identifying and selecting relevant papers to be reviewed on a particular topic and rigorously analysing the extracted data to answer a set of research questions [49]. The main components of our review protocol were: (i) research questions; (ii) search procedures; (iii) inclusion and exclusion criteria; (iv) papers selection; and (v) data extraction and data synthesis.

2.2.1 Research questions

The objective of this chapter is to identify and codify the most important quality attributes (RQ1) and architectural tactics (RQ2) for achieving those quality attributes in BDCA systems. Table 2.1 presents the research questions and their respective motivators.

Table 2. 1 Research questions of this SLR

Research question	**Motivation**
RQ1: Which are the most important quality attributes for BDCA systems?	To find out the most frequently emphasized quality attributes and the motivation behind their importance for a BDCA system.
RQ2: What are the architectural tactics for addressing quality concerns in BDCA systems?	To gain a detailed understanding of the various architectural tactics employed for achieving quality in BDCA systems.

2.2.2 Search strategy

According to the guidelines provided in [49] and [71], we defined a search strategy to retrieve as many relevant peer-reviewed papers as possible. Our search technique is described as follows.

2.2.2.1 Search method

We used automatic search for retrieving the potentially relevant papers from six digital libraries: ACM Digital Library, IEEE Xplore, Scopus, ScienceDirect, SpringerLink, and Wiley Online Library. These libraries were searched using the search terms introduced in Section 2.2.2.2. In order to ensure the identification and selection of as many relevant papers as possible, we used snowballing [72] for complementing the automatic search.

2.2.2.2 Search terms

We designed a search string (shown in Figure 2.1) according to the guidelines provided in [49], which consisted of two parts – security and big data processing frameworks. We initially included architecture-centric parts (i.e., quality attribute and architectural tactic), however, the search string returned only 14 papers. One possible reason (supported by Section 2.3.2) for the low number of papers can be that BDCA systems are primarily published outside Software Engineering (SE) venues, thereby, the papers do not explicitly use architecture-centric terms (e.g., quality attributes and architectural tactics). We ensured through our inclusion criteria (i.e., *I2*) that only the papers reporting architectural solutions were selected. For the two parts (i.e., security and big data processing frameworks), we incorporated the synonyms and the terms related to the basic two terms. We finalised the search string after ensuring that the pilot searches using the search terms were returning the known papers related to our review. The search terms in the string were matched only with the title, abstract, and keywords of the papers in the searched digital databases (except SpringerLink which does not allow to restrict

TITLE-ABS-KEY *((("security"* ***OR*** *"intrusion"* ***OR*** *"IDS"* ***OR*** *"SIEM"* ***OR*** *"anomaly detection"* ***OR*** *"outlier detection"* ***OR*** *"fraud detection"* ***OR*** *"network monitoring"* ***OR*** *"forensic"* ***OR*** *"threat"* ***OR*** *"data leakage"* ***OR*** *"data breach"* ***OR*** *"data theft"* ***OR*** *"data exfiltration"* ***OR*** *"attack"* ***OR*** *"data stealing")* ***AND*** *("hadoop"* ***OR*** *"HDFS"* ***OR*** *"mapreduce"* *OR* *"spark"* ***OR*** *"flink"* ***OR*** *"storm"* ***OR*** *"samza"* ***OR*** *"mesos")))*

Figure 2. 1 Search string for this SLR

the search to specific parts of a paper). We found that several of the retrieved papers were focussed on improving the security of big data and associated technologies (instead of leveraging big data and associated technologies for security, which is the focus of this review). We filtered out those papers during the inclusion and exclusion phase.

2.2.2.3 Data sources

Table 2.2 shows the searched digital libraries. The IEEE Xplore and ScienceDirect do not support the execution of a query with more than 15 terms. We had to split our query into two parts to make it run on IEEE Xplore and ScienceDirect. Apart from SpringerLink, we ran our query on other databases to match the terms only in title, abstract, and keywords. We have already mentioned that SpringerLink does not support searches in the specific parts of a paper [73]. The limitation of SpringerLink forced us to either limit the search only to the title of a paper or apply the string to the whole text of each of the potentially relevant papers. Whilst the former retrieved a very low number of papers, the later retrieved quite a large number of potentially relevant papers (8483 in total). To solve this issue, we followed the strategy used in [73]. According to this strategy, we examined only the first 1000 papers out of 8483 returned papers. This strategy does not pose a significant threat of missing papers as our data sources include Scopus, which encompasses a large number of papers indexed on SpringerLink. We did not use Google scholar due to its low precision of the results and the tendency of returning irrelevant papers.

Table 2. 2 Database sources

Source	URL
IEEE Xplore	http://ieeexplore.ieee.org
Scopus	https://www.scopus.com/
ScienceDirect	http://www.sciencedirect.com
ACM	http://portal.acm.org
SpringerLink	https://link.springer.com/
Wiley	http://onlinelibrary.wiley.com/

2.2.3 Inclusion and exclusion criteria

Table 2.3 shows the inclusion (I) and exclusion (E) criteria applied for selecting the papers. We targeted papers that highlighted the importance of one or more quality attributes and leverage, propose, or evaluate the incorporation of architectural solutions (architectural tactics) for achieving certain quality attributes (e.g., performance, reliability, accuracy, and scalability) in BDCA systems. Our search string returned the papers that were not architecture-based (e.g., algorithmic solutions or solutions based on controlling some operating parameters), we excluded such papers. We also excluded the papers that addressed the security of the big data itself or its associated technologies (e.g., the security of Hadoop). However, we made sure that any paper that leveraged big data technologies for the security of big data itself or its technologies should not be discarded from this SLR.

Table 2. 3 Inclusion and exclusion criteria

Inclusion criteria
I1: A study that is leveraging big data and its corresponding tools and technologies for cyber security
I2: A study that is architecture-based, which means the study should provide an architectural solution (i.e., components and architectural model) to the BDCA problem
I3: A study that emphasizes the importance of one or more quality attributes for BDCA
I4: A study that evaluates the proposed BDCA solution(s)
Exclusion criteria
E1: A study that is not peer-reviewed (e.g., position papers, panel discussion, editorials, and keynotes)
E2: A study written in a language other than English

2.2.4 Study selection

The papers selected from each digital database at each stage are shown in Figure 2.2. The different phases of the selection process are briefly described in the following.

- *Automatic search:* We ran our designed search string on six digital databases and retrieved a total of 4634 papers as a result of running automatic searches.
- *Title-based selection:* We read the title of the papers to quickly decide whether or not each of the retrieved papers was relevant to our SLR. In cases, where we were not able to decide about the relevance of a paper only based on its title, the paper was transferred to the next round of selection. Title-based selection reduces the pool of papers from 4634 to 748.
- *Duplication removal:* Scopus indexes papers available in several databases such as IEEE Xplore and ACM, therefore, it was expected that there will be duplicate papers in the pool of 748 selected papers. In this phase, we removed the duplicate papers, which reduced the number of our papers to a total of 516.
- *Abstract-based selection:* We read the abstract of each of the 516 papers to ensure that the papers were related to our SLR. During this stage, we discarded 348 papers that left us with 168 papers.
- *Full-text selection:* We completely read each of the 168 selected papers. A total of 69 papers were selected based on reading the whole text of the 168 papers.
- *Snowballing:* We used snowballing method [72] to explore the references of the 69 selected papers. We found 26 potentially related papers. Based on applying inclusion and exclusion criteria, we selected 5 papers from the 26 papers, which brought us to the final total of 74 papers for this SLR.

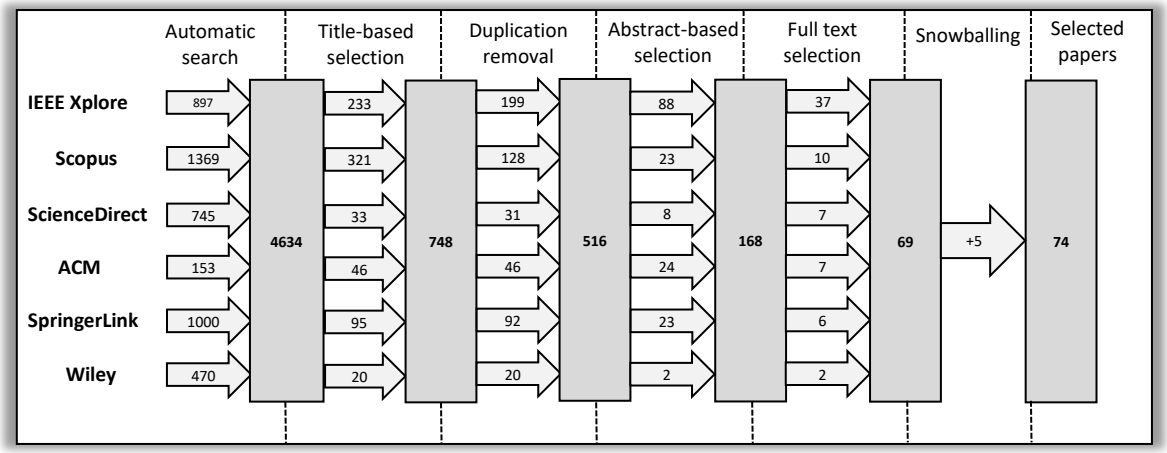


Figure 2. 2 Phases of study selection process

We did not restrict the selection based on the publication date of a paper. The reason for this was that the incorporation of big data technologies for cyber security is a comparatively new research field and as such our search process did not return the papers published before 2009. Appendix A enlists the papers selected for this SLR. We used the terms paper and study interchangeably in this thesis. Each paper has a unique identifier (S#) as presented in Appendix A. For example, the paper "A Cloud Computing Based Network Monitoring and Threat Detection System for Critical Infrastructures" is identified as S10. The system name included in Appendix A refers to the name of a BDCA system that was described in a paper. Some authors did not use an explicit name for their respective systems. For such systems, we followed the strategy adopted in [67] to name the system based on the unique features of each of the systems for discussing different aspects of those systems.

2.2.5 Data extraction and synthesis

This section describes the process of data extraction from the selected papers and the analysis of the extracted data to answer the research questions of this SLR.

2.2.5.1 Data extraction

We extracted the required data from the selected studies using a data extraction form that included the data items envisaged necessary to answer this review's research questions. The data extraction form is shown in Table 2.4. The data items D5 (publication venue) and D6 (citation count) were included as the indicators of the quality of the selected papers. The extracted data were recorded in an MS Excel spreadsheet for analysis.

2.2.5.2 Data Synthesis

We extracted three types of data – demographic data, quality attributes, and architectural tactics. We analyzed the demographic data using descriptive statistics. The results of our analysis of the demographic data have been presented in Section 2.3. We analyzed the data items D11 (quality attributes), D12 (rationale for quality attributes) and D13 (architectural tactics) using thematic analysis

Table 2. 4 Data extraction form

S.NO	Data item	Description	Research Questions
D1	Authors	Authors of the paper	
D2	Year	Publication year of the paper	Demographic data
D3	Title	Title of the paper	
D4	Publication type	The publication type of paper (e.g., conference)	Demographic data
D5	Publication venue	The venue where paper is published	Demographic data
D6	Citation count (google scholar)	The number of times paper is cited according to google scholar	
D7	Application domain	The type of BDCA system (e.g., Intrusion detection system)	Demographic data
D8	Data source(s)	The type of data the system is using for security analytics (e.g., network data)	Demographic data
D9	Big data processing framework	The processing framework system is leveraging (e.g., Hadoop)	Demographic data
D10	Proposed technique	A short summary of the security analysis technique proposed in the paper	
D11	Quality attribute(s)	The quality attribute(s) emphasized in the paper	RQ1
D12	Rationale for quality attribute(s)	The motivation for highlighting the quality attribute(s)	RQ1
D13	Architectural tactic(s)	The architectural tactic(s) proposed in the paper	RQ2

method [74], which is a widely used qualitative data analysis method that identifies themes extracted from multiple studies, interpret the themes, and draw conclusions. Our data analysis process is depicted in Figure 2.3. We explain our process step-by-step with the help of an example for one quality attribute (i.e., response time (Section 2.4.1)) and an associated tactic (i.e., Feature Selection and Extraction (Section 2.5.1.3)). The example illustrates the working from the raw data extracted from the primary studies to the reporting of quality attribute and codification of the associated tactic.

Quality attribute (RQ1): For answering RQ1, we identified the explicitly mentioned quality attributes in the selected papers as shown in Figure 2.3. For example, the response time quality attribute is identified through explicit indication of the significance of execution time, throughput, latency, or performance in the papers. After identifying the quality attribute, we extracted the reported motivators for a particular quality attribute in the reviewed system. We then applied the thematic analysis methods to the extracted motivators to become familiar with the data, generate initial codes, search the relevant themes, and name the themes. For instance, the motivators extracted from [S1], [S7], [S9], [S26], [S48] were themed as 'large size of data' as these papers argue that without high performance, a BDCA system will be overwhelmed with the massive amount of security event data. The application of the thematic analysis generalized and refined the extracted motivators as shown in Figure 2.3.

Architectural tactic (RQ2): For answering RQ2, we analysed the reported solutions to identify and record a brief definition of the tactics that were employed for achieving the desired quality attributes. The brief definitions were subjected to thematic analysis that enabled us to generalize, refine, and name the tactics. For example, the data extracted from [S9], [S10], [S12], [S13], [S14], [S15], [S31], [S32], [S33], [S55], [S57] mention that the selection and extraction of features from the collected data improves performance by discarding irrelevant features, which was themed as 'Feature Selection'. Once the tactic had been identified, we analysed each tactic from multiple perspectives. First, we identified and extracted the motivators for choosing the specific tactic for achieving the desired quality attribute. We then applied the thematic analysis method on the motivators extracted from multiple papers to get generalized and refined motivators for the tactic. For example, the papers [S9], [S10], [S14], [S32], [S33], [S55] urge that several features within the collected security event data are irrelevant with respect to attack detection, hence, selection and extraction of pertinent features can help improve performance.

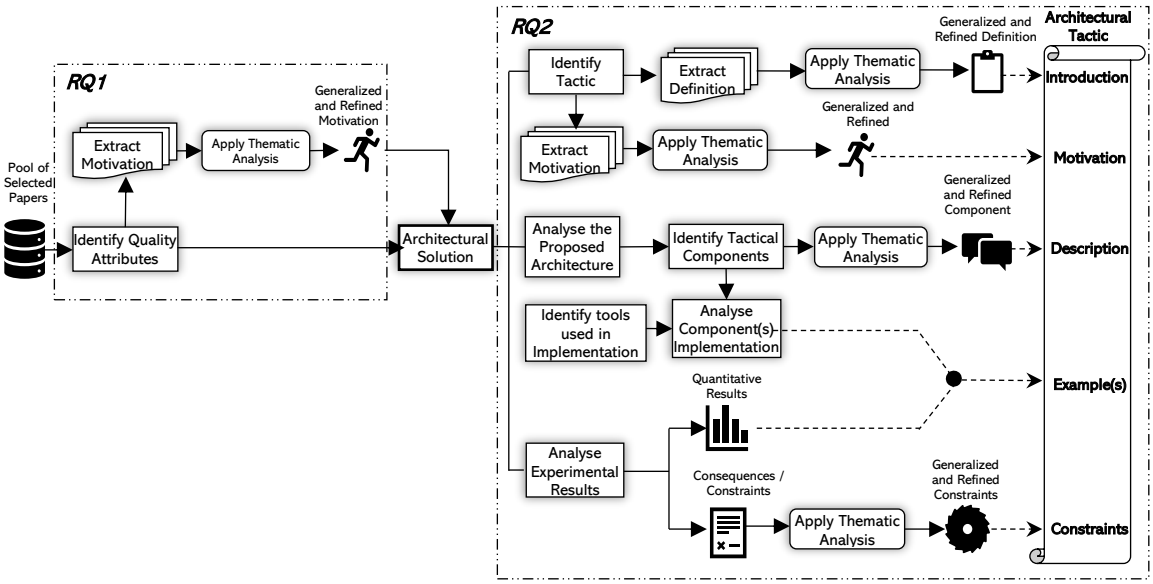


Figure 2. 3 An overview of our data analysis process

We themed this motivator as 'irrelevancy of features' that is one of the motivators for using Feature Selection tactic. Next, we investigated the proposed architecture to identify the components related to each of the identified tactics and their interactions with other components in the reported architectures. This was followed by the application of thematic analysis to popularize the components that best capture each tactic's details and their interactions. In the case of Feature Selection tactic, the identified components include Feature Selection and Feature Extraction that interact with data storage and data processing components respectively.

We analysed the implementation aspects of the identified components and the tools used for implementation. This aspect of the tactics is reported in the example subsection. We analysed the experimental results related to each tactic as reported in the reviewed papers. The quantitative results were reported along with examples while the consequences of each tactic, as identified through experimental evaluation or elsewise, were extracted and subjected to thematic analysis as shown in Figure 2.3. For instance, several papers [S10], [S12], [S14] report that discarding important features can be detrimental to accurately detecting attacks, which is themed as 'discarding important features' and is reported as one of the constraints for the Feature Selection tactic. After codifying all the tactics, the dependencies among the tactics were identified based on our analysis and reflections.

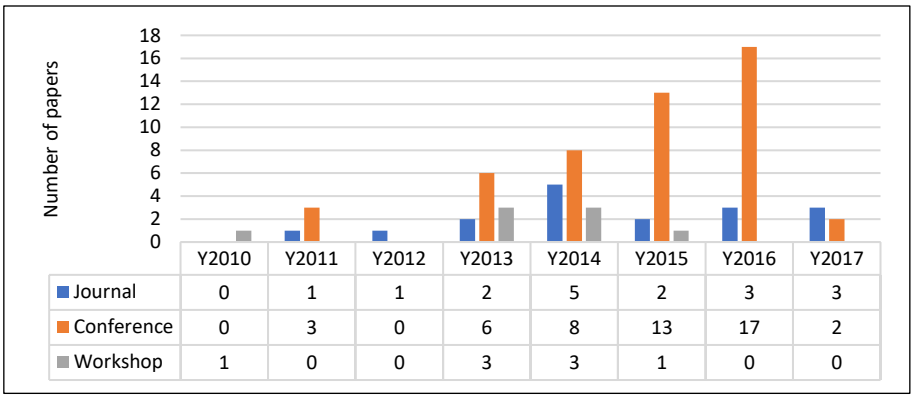
Our data synthesis process revealed that the architectural aspects of BDCA systems are primarily discussed around 12 main themes (i.e., quality attributes), which are shown in Table 2.6 along with the papers belonging to each theme. As shown in Figure 2.3, the thematic analysis has been applied at multiple levels of abstraction (e.g., quality attributes and architectural tactics), which enabled the analysis process to reveal a list of sub-themes (i.e., architectural tactics). These sub-themes, shown in Figure 2.8, are associated with the main themes (i.e., quality attributes). The themes and sub-themes have been elaborated in Section 2.4 and Section 2.5 respectively.

2.3 Meta-data Analysis

This section reports the distribution of the reviewed papers along the years and types of publications, big data processing frameworks used, application domains, and data sources leveraged by the reviewed BDCA systems.

2.3.1 Chronological view

Figure 2.4 shows that the reviewed papers were published between 2010 and 2017. Our SLR covers the papers published before *8th May 2017* when the search process for selecting the potentially relevant papers was completed. Figure 2.4 shows a regular upward growth in the number of papers on security



	Y2010	Y2011	Y2012	Y2013	Y2014	Y2015	Y2016	Y2017
Journal	0	1	1	2	5	2	3	3
Conference	0	3	0	6	8	13	17	2
Workshop	1	0	0	3	3	1	0	0

Figure 2. 4 Number of selected papers per year and their distribution over venues' types

analytics leveraging big data technologies. This is in accordance with the ideas surfaced in the Cloud Security Alliance Workshop [75] where it was anticipated that the role of big data analytics in security will continue increasing. This upward trend can be interpreted from two perspectives. First - the threat landscape is changing from cyber attacks launched by individuals to cyber attacks launched by organized groups. Second - the traditional tools and technologies are unable to deal with the high volume, large size, and heterogeneous nature of security events data. Therefore, the adoption of big data technologies is receiving an increasing attention from the cyber security researchers.

2.3.2 Publication venues and types

Figure 2.4 indicates that most of the papers (49 papers (66.2%)) have been published in conferences; there were 17 (22.9%) journal papers and 8 (10.8%) workshop papers. The 74 reviewed papers were published in 59 venues, in which BigData Congress, Journal of Supercomputing, and Security and Privacy in Big Data workshop are the leading venues for publishing work on security analytics. These three venues have published three papers each. Five venues have published two papers each. These venues include TrustCom/BigDataSE/ISPA, Conference on Systems, Man, and Cybernetics, Network Operations and Management Symposium, Special Interest Group on Data Communication (SIGCOMM), and Conference on Parallel and Distributed Systems. The rest of the 55 papers were published in 55 different venues. The selected papers have been primarily published in three research areas – Big Data (21 papers), Network Communications (17 papers), and Cyber security (15 papers). It is interesting to note that only four papers have been published in the Software Engineering related venue. These findings show that researchers with different research backgrounds are interested in BDCA.

2.3.3 Big data processing frameworks

We have classified the reviewed papers based on the big data processing framework employed in the BDCA systems. The processing framework provides guidelines for processing big data. In the reviewed papers, there are three frameworks used – Hadoop[4] (49 papers), Spark[5] (14 papers), and Storm[6] (10 papers). These frameworks are being used by well-known organizations such as Yahoo, Google, IBM, Facebook, and Amazon [76]. Figure 2.5 shows this information. We could not classify one of the papers into either of the three categories based on the available information. Figure 2.6 shows the patterns of the use of these frameworks adopted over the years. It is interesting to note that in the

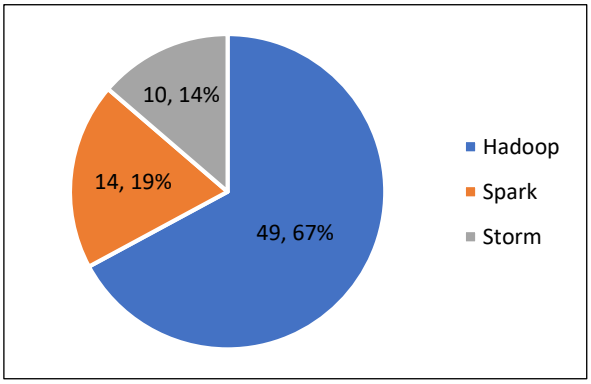


Figure 2. 5 Number and percentage of papers using various big data frameworks

[4] http://hadoop.apache.org/
[5] https://spark.apache.org
[6] http://storm.apache.org/

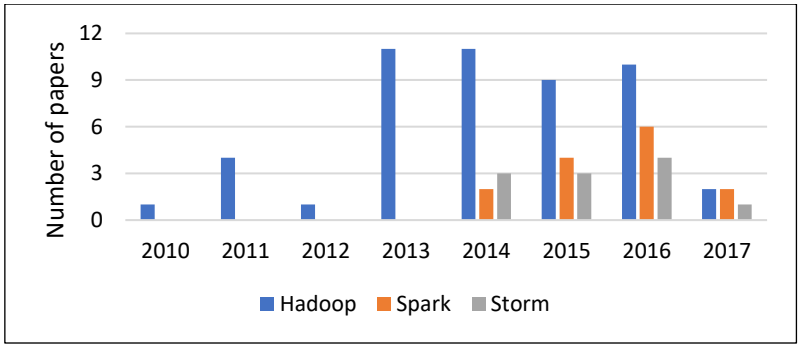


Figure 2. 6 Yearly distribution of papers over type of big data frameworks

first four years (i.e., from 2010 to 2013), only Hadoop has been incorporated whilst for the last four years (i.e., from 2014 to 2017), a steady upward trend can be observed for adopting Spark and Storm. One possible reason for this trend can be the migration from batch processing to stream processing due to the rapidly changing time-sensitive requirements of BDCA systems. The low numbers shown in 2017 is due to the reason that the papers published only before 8th May 2017 are included in 2017.

2.3.4 Application domains

We have categorized the included papers based on the application domain. For such categorization, we have analysed the data item D7 in the data extraction form (Table 2.4). The domain-based categorization is of potential value for researchers and practitioners interested in the domain-oriented prospects of BDCA systems. The papers on BDCA systems are primarily of two types – generic and specific. The generic category reports BDCA systems (e.g., Intrusion Detection System (IDS) and Alert Correlation) for detecting a variety of attacks. The specific category includes the papers that report BDCA systems designed for detecting a particular type of attack such as Denial of Service (DoS). There are several cases where an IDS is evaluated with a particular attack (e.g., [S4] and [S12]), however, we have included such systems in the IDS category only. As shown in Table 2.5 that the 74 reviewed papers have been categorized into nine groups. The majority of the reviewed papers belong to IDS category (i.e., 34 papers) followed by unclear (11 papers) and alert correlation (i.e., 8 papers). The unclear category includes the papers that cannot be explicitly (i.e., without authors' interpretation) categorized into either of the remaining 8 groups.

2.3.5 Data sources

A BDCA system can collect data from several data sources, which have been classified into three categories in this study – (1) *Network,* (2) *Host,* and (3) *Hybrid*. Figure 2.7 shows the number and the percentage of papers belonging to each category. The systems belonging to the first category (i.e., network) collect data from the network infrastructure of an organization. Within the network infrastructure, a BDCA system can leverage different kinds of data such as NetFlow data, packet data, honeypot data, IDS log data, and firewall logs [77]. The systems belonging to the second category (i.e., host) collect data from an organization's host machines. These data sources include but not limited to operating system logs, system call logs, web server logs, email logs, and windows event logs. The systems belonging to the third category (i.e., hybrid) collect data from both network and host machines. Figure 2.7 shows that most of the systems (i.e., 55) in our review rely on the data collected from the network, followed by hybrid (i.e., 13), and only 6 systems leverage host data for security analytics. One possible reason for such a focus on the network-based BDCA systems could be that unlike host-based systems that protect only a specific host within an organization, the network-based systems take into

Table 2. 5 Distribution of the application domains of the reviewed papers

S. No	Application domain	# of papers	Papers
1	Intrusion Detection System	34	S1, S3, S4, S5, S8, S9, S12, S14, S15, S18, S21, S26, S27, S29, S31, S32, S34, S35, S41, S44, S46, S48, S51, S54, S55, S57, S59, S63, S64, S66, S67, S72, S73
2	Alert Correlation	8	S6, S7, S11, S16, S19, S38, S39, S65
3	DoS Detection	7	S2, S20, S28, S33, S43, S61, S62
4	Botnet Detection	4	S22, S23, S36, S69
5	Forensic Analysis	2	S30, S53
6	APT Detection	2	S37, S47
7	Malware Detection	4	S43, S45, S60, S68
8	Phishing Detection	2	S17, S50
9	Unclear	11	S10, S13, S25, S40, S49, S52, S56, S58, S70, S71, S74

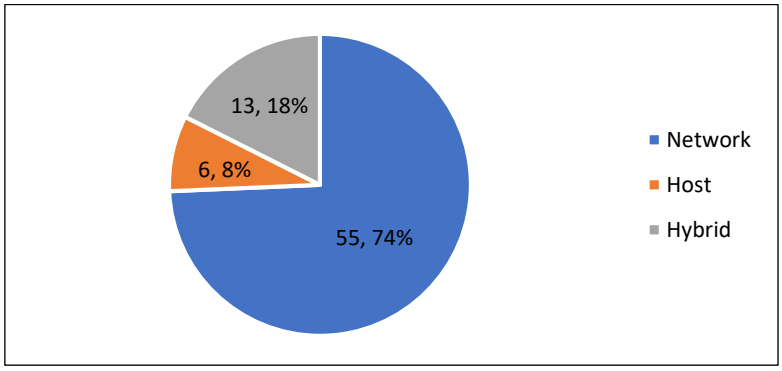


Figure 2. 7 Number and percentage of papers distributed over data source

consideration the security of an entire organization's network. The pros and cons of network-based and host-based security systems have already been well explored. Interested readers can refer to [78-80].

2.4 RQ1: Quality Attributes for BDCA Systems

This section reports the results based on the analysis of the data about the critical quality attributes reported for BDCA and their respective rationale. The analysis is meant to answer RQ1, "*Which are the most important quality attributes for BDCA systems?*". We answer this question from two perspectives: (1) We present the statistical analysis of the quality attributes identified in the reviewed papers to understand the depth of the emphasis placed on the identified quality attributes and (2) We report the identified motivation to justify that why these particular quality attributes have been highly emphasized for BDCA.

2.4.1 Statistics of the quality attributes

This section reports the results from the analysis of the data for item D11 of the data extraction form (Table 2.4). The item D11 records the information about the quality attributes emphasized in the reviewed papers. Table 2.6 presents the number, percentage, and identifiers of papers emphasizing particular quality attributes. These are the quality attributes that have been emphasized, highlighted, or considered important for BDCA systems. If a paper reports a quality attribute important that does not mean that that paper also focuses on achieving that particular quality attribute. For example, one paper [S15] highlights interoperability as a critical quality attribute for BDCA systems, however, it does not report any strategy for achieving interoperability. Multiple attributes may be emphasized in one

Table 2. 6 Papers emphasizing various quality attributes for BDCA systems

Quality attribute	# of papers	% of papers	Paper identifiers
Response Time	67	90.5	S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S18, S19, S20, S21, S23, S24, S26, S27, S29, S30, S31, S32, S33, S34, S35, S36, S37, S38, S39, S40, S41, S42, S43, S44, S45, S46, S47, S48, S49, S50, S51, S52, S53, S54, S55, S56, S57, S58, S60, S61, S62, S63, S64, S65, S66, S67, S70, S71, S73, S74
Accuracy	43	58.1	S1, S2, S5, S8, S9, S10, S12, S14, S15, S18, S20, S21, S22, S23, S24, S25, S26, S27, S28, S29, S30, S31, S36, S37, S40, S41, S42, S43, S46, S47, S49, S55, S57, S58, S59, S60, S61, S62, S64, S66, S67, S70, S73
Scalability	40	54.0	S2, S3, S5, S6, S7, S8, S9, S10, S12, S15, S16, S17, S18, S19, S21, S22, S24, S25, S29, S30, S31, S38, S42, S45, S47, S48, S49, S51, S52, S53, S54, S56, S57, S61, S65, S66, S68, S70, S73, S74
Reliability	23	31.0	S2, S6, S8, S10, S13, S17, S18, S19, S20, S30, S31, S43, S48, S49, S52, S54, S55, S59, S61, S62, S65, S66, S71
Usability	18	24.3	S10, S17, S19, S24, S29, S30, S40, S41, S51, S52, S53, S56, S65, S66, S68, S72, S74
Interoperability	15	20.2	S10, S15, S16, S17, S19, S35, S39, S40, S42, S46, S47, S50, S51, S54, S72
Adaptability	11	14.8	S1, S7, S10, S15, S20, S28, S28, S39, S42, S61, S74
Modifiability	7	9.4	S7, S10, S12, S29, S40, S44, S56
Generality	7	9.4	S7, S10, S12, S19, S22, S29, S51
Privacy assurance	7	9.4	S15, S19, S25, S30, S36, S42, S49
Security	4	5.4	S10, S49, S53, S73
Stealthiness	3	4.05	S9, S21, S51

paper such as [S2] reports response time, accuracy, scalability, and reliability quality attributes for a BDCA system.

Table 2.6 shows that response time (67 papers), accuracy (43 papers), and scalability (40 papers) have been reported as important quality attributes for BDCA by several papers. The limited emphasis (i.e., only 4 papers) on security as a quality attribute warrants to distinguish between two scenarios – (i) a software system being itself secure (security as a quality attribute [81]) and (ii) a software system (e.g., BDCA system) providing security to an organization's cyberspace (security as a functional attribute [81]). All of the 74 reviewed papers consider the security of the cyberspace of an organization important. However, only 4 papers explicitly consider and address the security of BDCA system. Apart from the reported 12 quality attributes, the reviewed papers also mentioned some other quality attributes, which were found in less than 3 papers are not shown in Table 2.6. These quality attributes include design simplicity [S6], algorithm expressiveness [S6], modularity [S10], uniform programmability [S17], and reusability [S74].

2.4.2 Quality attributes definition for BDCA systems

This section reports our analysis of the data item D12. The objective is to clarify the definition, use, and emphasis on the 12 quality attributes presented in Table 2.6. Such clarification will help answer the question that why these particular quality attributes are more important for a BDCA system. For each quality attribute, we provide a definition and motivation for their achievement in a BDCA system.

Response Time is a measure of how quickly a system responds to an event [81]. Response time is characterized by different properties such as execution time, throughput, and latency. Response time is a highly critical attribute of BDCA systems [S1], [S17], [S20], which are expected to respond in real-time. It can be difficult for BDCA systems to achieve the desired level of real-time response as the security event data is usually huge [S1], [S7], [S9], [S26], [S48] and is generated at a very high speed [S3], [S46]. The collected data need to be pre-processed to ensure data quality. The data dimensionality

needs to be reduced [S44] and the data should be converted from binary format to text format [S16]. The current threat landscape requires a widespread correlation and contextualization of security event data to detect sophisticated attacks designed to execute over a period of time. Such threats need complex computation that can make real-time analytics a challenging task [S19], [S26].

Accuracy is a measure to which a system provides correct results with the needed degree of precision [82]. To make it specific to security analytics, it is the ratio of the total number of correctly detected attacks to the total number of attacks [S31]. As evident from Table 2.6, accuracy is a highly critical quality of BDCA systems. A failure in detecting an attack can lead to catastrophic outcomes [83]. An accurate BDCA system is expected to detect and resist attacks without thwarting legitimate access requests (i.e., false positives) [S1], [S46].

Scalability is a measure of how easily a system can grow to handle more user requests, transactions, servers, or other extensions [81]. Table 2.6 reveals that around 54% of the reviewed papers consider scalability as an important quality attribute for BDCA systems, which are expected to be highly scalable because the volume and velocity of security event data are quite unpredictable [84][S7]. BDCA systems analyse constantly growing size of security event data to detect and thwart sophisticated cyber attacks (e.g., APTs) [S17], [S31], [S74]. For example, the Operation Shady RAT attack started in 2006 and was detected in 2011 when it had affected around 72 different organizations worldwide. Organizations are forced to monitor more and more sources (e.g., database access and users' activities) to deal with sophisticated attacks [S56]. Some of the reported challenges in achieving scalability include: (a) inefficient communication among a large number of processors in a distributed setup [S57]; (b) unfair load-balance among the computing nodes [S57] and (c) the choice of centralized data storage [S50].

Reliability is a measure of how long a system runs before experiencing a failure [81]. There are 23 reviewed papers (Table 2.6) that highlight the importance of reliability for BDCA systems, which may be vulnerable to a number of failures such as a data processing node can crash because of large data size [S1], [S18]. Reliable data collection is also quite crucial for security analytics [S33]. By reliable data collection, we mean that the data collector should cope with the speed of data so that all security event data (e.g., NetFlow data [85]) can be captured. It is quite possible that a single NetFlow may contain information significant for detecting an attack and letting such a NetFlow go uncaptured and unprocessed means letting the attack go undetected. It is worth mentioning that the traditional data collectors (e.g., Wireshark) are lagging behind in efficiently collecting data under peak conditions (e.g., DoS attack) [S63]. The inherent design limitations of software and hardware also need some attention while designing a BDCA system as it may lead to several types of malfunctioning [S71].

Usability is a measure of how easy it is for people to learn, remember, and use a system [81]. Considering the spontaneous nature of cyber attacks, where delaying the response by a few seconds can be consequential, it is important that a BDCA system is user-friendly [S10]. A BDCA system should provide user-friendly functionality to visualize security incidents (i.e., threat alerts) so that the required mitigation action can be taken [S17], [S29], [S41], [S58], [S66]. Such a system should also incorporate some mechanism that enables a security administrator to focus on the most dangerous alerts [S41] out of hundreds of alerts.

Interoperability is a measure of how easily a system can interconnect and exchange data with other systems or components [81]. Table 2.6 shows that around 20.2% of the reviewed papers consider interoperability as an important quality attribute for BDCA systems. Unlike other data-intensive systems, the interoperability requirement is more critical for such systems, which need to be integrated together to enhance the overall security spectrum of an organization [S14], [S17]. For example, a

collaborative IDS enables multiple IDSs to collaborate with each other for detecting sophisticated attacks [86]. A BDCA system connects to a variety of sources (e.g., network devices and host machines) for data collection for which limitation on interoperability can be problematic. Motivated by its data-intensive nature, a BDCA system needs to interoperate with a variety of databases such as Oracle, MySQL, and MsSQL [S10], [S16], [S47]. Moreover, a BDCA system should also be able to integrate with specialized hardware (e.g., GPU) for achieving fast data processing capability [S35].

Adaptability is the measure of how easily a system adapts itself to different specified environments using only its own functionality [82]. It is expected that a BDCA system automatically adjusts itself to various kinds of changes as apparent from our findings shown in Table 2.6. A BDCA system incorporating multiple data sources deals with different data formats so the system should be able to automatically adjust itself with data format without affecting the overall performance of a system [S1], [S39], [S74]. Moreover, a system should automatically change the security policies according to the requirements [S10]. For example, more in-depth monitoring of insiders during working hours as compared to non-working hours. Similarly, the network of an organization frequently experiences changes (e.g., change in network topology), therefore, a BDCA system should adjust itself to the changes in a network environment [S15].

Modifiability is a measure of how easy it is to maintain, change, enhance, and restructure a system [81]. Modifiability relates to manual modification by a user while adaptability is the automatic adjustment by a system itself without any involvement of a user. Several changes are beyond the control of a system itself and require input from a user. In the context of security analytics, a number of different modifications are required from time to time. A signature-based BDCA system needs to be updated with the latest and emerging attack patterns [S7]. Similarly, with the advancement of algorithms and computational models, a system needs to be modified to employ the advanced algorithms and models [S10], [S12], [S56]. A BDCA system should be flexible enough to easily incorporate the upcoming tools and technologies [S17], [S40].

Generality is a measure of the range of attacks covered by a security system [S7]. Table 2.5 indicates that different BDCA systems target the detection and prevention of different types of attacks. It is also mentioned in the application domains (i.e., Section 2.3.4) that some of the BDCA systems (e.g., IDS, alert or correlators) are generic and cover a wide variety of attacks. BDCA systems are designed by considering a trade-off between generality and specificity; the latter types of systems are more accurate [87] but less flexible to deal with different types of attacks. Several of the reviewed papers ([S7], [S12], [S19], [S22]) assert that a BDCA system has to be generic enough to counter the variety, complexity, and sophistication of different types of attacks. If an organization's security experts believe that their organization is vulnerable only to a particular type of attack (e.g., phishing), then a specific security solution should be preferred over a generic solution.

Privacy assurance is the measure of the ability of a system to carry out its business according to defined privacy policies [88]. Several of the reviewed papers ([S15], [S19], [S25], [S30], [S36], [S42], [S49]) highlight the importance of privacy assurance for security analytics. In the context of BDCA, privacy assurance is quite critical because a number of BDCA systems capture, store, and process packet payload, which contains personal data of users. For example, one of the reviewed papers i.e., [S17] employs a technique called content inspection [89], which in addition to the header information, also capture, store, and process payload of network packets. According to the European data laws and regulations [90], it is not allowed to store and process the packet payload without the relevant users' consent.

Security is the measure of how well a system protects itself and its data from unauthorized access [81]. A BDCA system must also itself be secured to ensure the security of an organizational IT infrastructure. Otherwise, such systems can be hacked for modifying the monitoring rules or the data transitioning between two services (e.g., data collection and data processing). Such unauthorized modifications can negatively affect a security system's abilities to safeguard organizational IT infrastructure; such have been highlighted in [S10], [S49], [S53], [S73].

Stealthiness is the measure of a system's ability to function without being detected by an attacker [87]. If an attacker can figure out the whereabouts of a security system, the attacker is very likely to first focus on disabling the security system. It is important that a BDCA system operates in a stealthy mode, whereby an attacker does not even know that a security system is in place for monitoring an attacker's activities. In our review, only three papers (i.e., [S9], [S21], [S51]) highlight the significance of stealthiness for a BDCA system.

2.5 RQ2: Architectural Tactics for BDCA Systems

This section reports the architectural tactics for BDCA systems discovered from the reviewed papers. This part of our study is meant to answer RQ2, "*What are the architectural tactics for addressing quality concerns in BDCA systems?*" We have identified the key components of a BDCA system and arranged them in a way to form a so-called reference architecture so that the details of each of the codified tactics can be reported and understood in the context of a BDCA system's architecture. We also present the template used for codifying the tactics.

BDCA Reference Architecture: A Software Reference Architecture (SRA) captures the essence of architectures of similar systems (i.e., systems belonging to the same class or domain) [91]. An SRA helps increase the overall understanding of a particular class of systems (e.g., BDCA systems) and provides guidelines for developing a concrete architecture. The elements (e.g., tools, modules, and tasks) shown in the BDCA SRA (Figure 2.8) have been identified from the concrete architectures of the BDCA systems reported in the reviewed papers. The various elements are presented in Figure 2.8 at an abstract level, which make them applicable to any big data system. For example, the workflow of data analysis (i.e., data collection, data pre-processing and so on) is similar for all big data systems. However, these abstract elements are represented in more concrete terms within the description of the tactics. It is also to be noted that the various elements employed in a BDCA system architecture are not limited to what is shown in Figure 2.8. This SRA is reported as a diagrammatic guide to help understand the key role of each tactic in a typical architecture of a BDCA system. The representation for various elements (e.g., green rectangle for data collection module) is separately shown in the legend in Figure 2.8. We follow the same representation for reporting each of the tactics, so a reader can relate each tactic to the SRA, whose elements are explained below.

Data Sources include the sources that generate rich data, which can be collected and analysed for detecting and preventing cyber attacks. There are several options [92] (e.g., windows logs, NetFlow data, and email logs) that generate valuable data for security analytics. The choice of the source(s) varies from enterprise to enterprise.

Data Collection interfaces a system with external data sources. This module uses various tools [92] (e.g., Wireshark and Gulp) to collect data from the specified data sources and stores it in a Data Storage in its entirety.

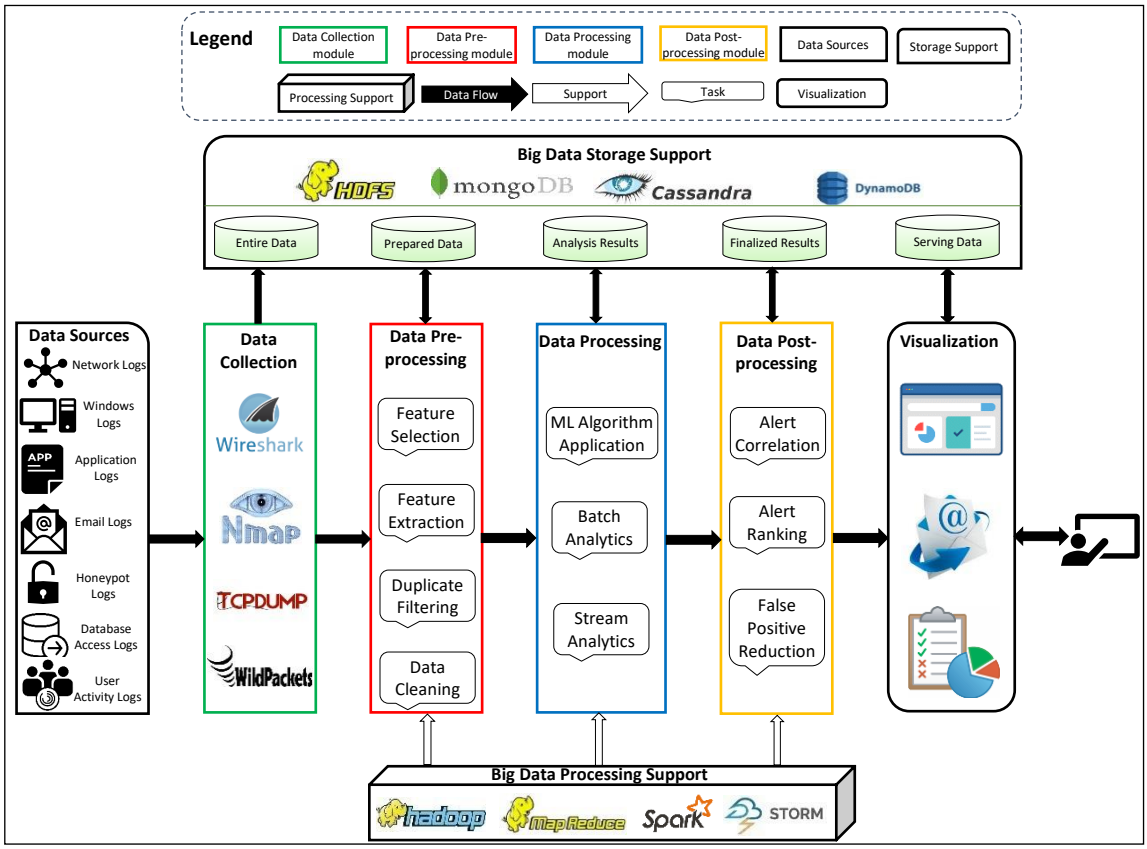


Figure 2. 8 Reference architecture for BDCA systems

Data Pre-processing prepares data for subsequent modules, especially data processing. The pre-processing can include several tasks such as feature selection and extraction, removal of duplicates and bad records, data validation and standardization. The data is received from the Data Storage and prepared data is stored back in the Data Storage as shown in Figure 2.8.

Data Processing module leverages big data technologies for extracting valuable insight about cyber attacks. It should be noted that big data technologies (e.g., Hadoop, Spark, or MongoDB) are not limited to what is shown in Figure 2.7 rather we direct a reader to [93] for the details.

Data Post-processing module improves the results generated by the Data Processing module by applying various techniques such as correlating and ranking alerts. Like Data Processing, this module communicates with the Data Storage and is supported by the big data technologies for the realization of the associated tasks.

Visualization module leverages various tools (e.g., dashboard, text or graphics report, and email notifier) to communicate the finalized results (e.g., security alerts) to security experts.

Big Data Storage Support manages the distribution and back and forth storage of security data after being processed in each module. The module not only manages data storage but also maintains various procedures followed by different modules to access and transform data. The module uses different data storage tools (e.g., HDFS, MongoDB, Cassandra) to support the required data storage.

Big Data Processing Support manages the distribution of data processing among computing nodes. The module uses big data framework (e.g., Hadoop, Spark, or Storm) to distribute data processing. As shown in Figure 2.8, the module supports data pre-processing, data processing, and data post-processing modules to distribute the processing among the computing nodes.

Tactics Template: The tactics have been codified using the following template.

- *Introduction:* a brief explanation of how the tactic achieves the desired quality attribute;
- *Motivation:* the rationale behind why the tactic needs to be incorporated in an architecture design;
- *Description:* explanation of how various system components interact to achieve the desired quality attribute and the architecture diagram highlighting the components related to a tactic;
- *Constraints:* necessary conditions for incorporating the tactic in the architecture of a system;
- *Example:* systems from the reviewed papers demonstrating the application of the tactic;
- *Dependencies:* whether or not the tactic depends upon other tactic(s);
- *Variation (optional):* slightly modified form of the original tactic;

Unlike [33], the codification of our tactics includes the details about constraints, examples, and dependencies. This is because our codified tactics are identified and extracted from the literature (similar to [67] and [68]), which enabled us to extract and report such details. We assert that the inclusion of constraints, examples, and dependencies in a tactic's description are quite useful in understanding the impact and limitations of each tactic. Given we use the same diagrammatic style for reporting all the tactics, we have to modify the diagrams in a way that can best explain the specific tactic instead of explaining the entire architecture of a BDCA system. For example, the component responsible for feature selection is not explicitly shown in tactics other than the Feature Selection tactic (Section 2.5.1.3). It does not mean that this component does not necessarily exist in the diagrams for other tactics. Similarly, the rationale behind showing the data collection as a distributed process (in Data Ingestion Monitoring tactic (Section 2.5.4.1) and Secure Data Transmission tactic (Section 2.5.5.1)) and as a centralized process is to suit the tactic and explain its key points. Furthermore, we use the term component instead of module (as used in SRA that partitions the system into implementation units [33]) during the description of tactics. This is because we want to illustrate the runtime interactions among the architectural elements for accomplishing various tasks.

Codified Tactics: An architectural tactic is a design strategy that influences the achievement of a quality attribute [33]. Figure 2.9 shows the codified tactics associated with various quality attributes. We have identified and codified six tactics for performance, four for accuracy, two for scalability, three for

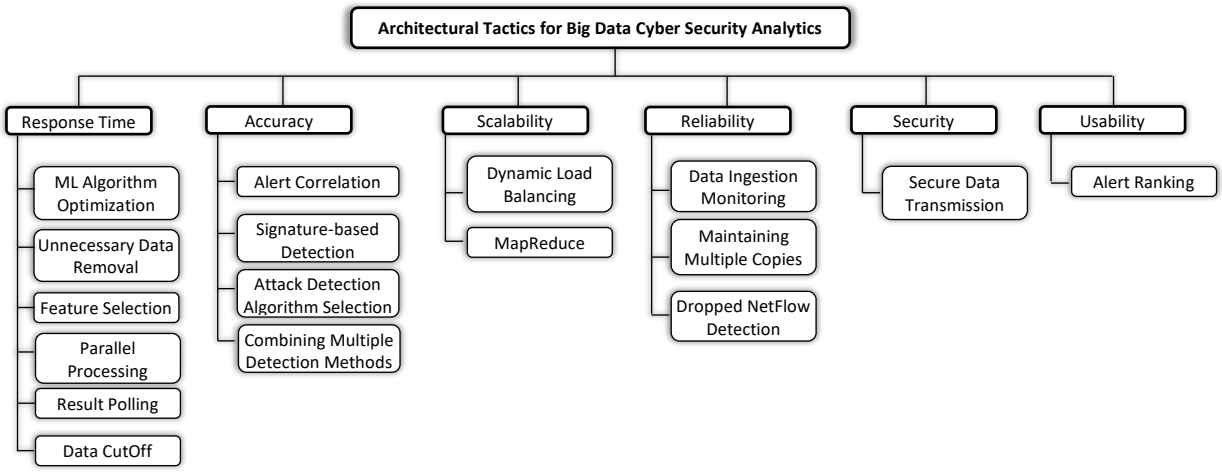


Figure 2. 9 Architectural tactics for BDCA Systems

reliability, and one for security and usability each. The codified tactics can be classified based on various criteria such as infrastructure, big data frameworks, ML pipeline, and quality attributes. Since the study reported in this chapter is focussed on quality attributes, we classified the tactics based on quality attributes as shown in Figure 2.9.

2.5.1 Response time

This section reports the architectural tactics related to Response time quality attribute.

2.5.1.1 ML Algorithm Optimization

Introduction. The ML Algorithm Optimization tactic has been found in all the papers as all of the BDCA systems leverage Machine Learning (ML) algorithm for analysing the security event data. This tactic guides the selection of a suitable algorithm for achieving computational efficiency for improving a system's response time.

Motivation. The two most important factors related to the response time of BDCA are input data type and ML algorithm [59]. The ML algorithms range from supervised learning (e.g., Logistic Regression, Support Vector Machine, Naïve Bayes, Random Forest, and Decision Trees) to unsupervised learning algorithms (e.g., K-means and Neural Networks) [59]. An ML algorithm can be selected based on several factors such as time complexity, incremental update capability, offline/online mode, and generalization capacity of the algorithm, and the potential impact of the algorithm on the detection rate (accuracy) of a system.

Description. The main components of ML Algorithm Optimization tactic are shown in Figure 2.10[7]. The *data collection* component collects security event data for training a BDCA system. The training data can be collected from sources within an enterprise as depicted in Figure 2.10 or an already available dataset such as KDD. The *data preparation* component prepares the data for training the model by applying various filters. The selected *ML algorithm* is applied to the prepared training data to train an attack detection model. The time taken by the algorithm to train a model (i.e., training time) varies from algorithm to algorithm. Once the model is trained, it is tested to investigate whether the model can

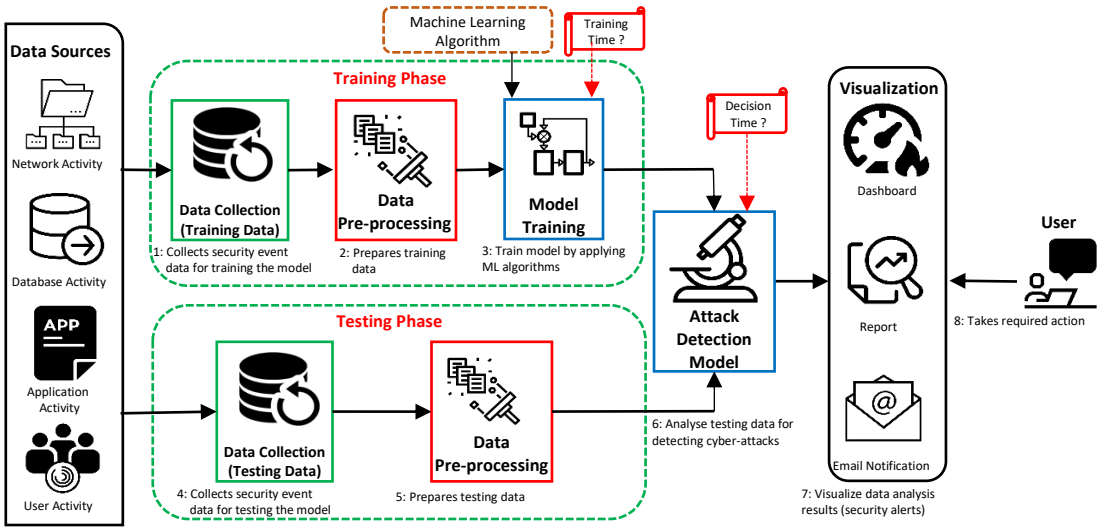


Figure 2. 10 ML algorithm optimization tactic

[7] Source of icons in the diagrams: https://thenounproject.com/

detect cyber attacks. For testing the model, data is collected from an enterprise as shown in step 4. The testing data is filtered through the *data preparation* component and fed into the *attack detection model*, which analyses the data for detecting attacks. The time taken by an *attack detection model* to decide whether a particular stream of data pertains to an attack (i.e., decision time) depends upon the employed algorithm. The result of the data analysis is displayed to the user through *visualization* component.

Constraint. Several things need to be considered while selecting and using an ML algorithm.

- An ML algorithm's response time may not be consistent in different domains [94]. Therefore, an ML algorithm may perform well for one type of security analytics (e.g., detecting DoS attack) but may not perform well in another type of security analytics (e.g., detecting brute force attack).
- The selection of an algorithm is challenging in the sense that in addition to response time, it affects other system's qualities such as accuracy, complexity, and understandability of the result. For example, Cheng et al. [95] compare SVM with Extreme Learning Machine (ELM) in terms of accuracy and response time. It is observed that SVM generates more accurate results but is computationally expensive. On the other hand, ELM generates less accurate results but is more light weight. A reasonable trade-off should be established among various system's qualities while selecting an algorithm.
- The selection of algorithm also depends upon the working mode (online or offline) of BDCA system. An algorithm having a time complexity less than $O(n^3)$ are considered acceptable for online mode while algorithms with a complexity of $O(n^3)$ and above are slower and suits only offline analysis mode [59].

Example. As mentioned, all our included systems leverage ML algorithms. We report the findings from a couple of papers to exhibit the role of an optimized algorithm in improving the response time of a BDCA system.

- Spark-based IDS Framework [S9]: This BDCA system compares the training time and decision time of five ML algorithms namely Logistic regression, Support vector machine, Random forest, Gradient boosted decision trees, and Naïve Bayes. With KDD dataset, Naïve Bayes shows the best training time (i.e., 79.5 sec) and SVM shows the worst training time (i.e., 479.12 sec). On the other hand, with respect to prediction time, SVM shows the best prediction time (i.e., 10 sec) and Gradient boosted decision tree shows the worst prediction time (i.e., 22.2 sec).
- Ultra-High-Speed IDS [S14]: Here, the BDCA system is tested with six ML algorithms to investigate the training time and decision time of each algorithm. The algorithms are Naïve Bayes, SVM, Conjunctive rule, Random Forest, J48, and RepTree. It is found that both in terms of training time and prediction time RepTree is the most efficient followed by J48.
- Cloud-based Threat Detector [S10]: The system has been implemented with two ML algorithms – K-means and Naïve Bayes, to explore the training time taken by both algorithms. It is observed that with 500 GB, K-means takes around 60 secs while Naïve Bayes takes around 92 secs to train the model.

Dependencies. ML Algorithm Optimization tactic requires Unnecessary Data Removal tactic (Section 2.5.1.2) and Feature Selection and Extraction tactic (Section 2.5.1.3) to help bring collected data into a refined form. After the application of these tactics, ML Algorithm Optimization tactic can be efficiently

applied to the refined data to quickly train a system and detect attacks. ML Algorithm Optimization tactic needs to be incorporated along with Attack Detection Algorithm Selection tactic (Section 2.5.2.3) as these two tactics establish the trade-off between the effects of ML algorithm on response time and accuracy.

2.5.1.2 Unnecessary data removal

Introduction. Unnecessary Data Removal tactic has been found in four of the reviewed papers ([S52], [S58], [S66], [S67]). This tactic removes the unnecessary data from the dataset of security event data that is supposed to be processed by the data processing component of a system to detect cyber attacks. The subset of security event data that does not contribute to the detection process is termed as unnecessary data. The removal of such data from the dataset reduces the size of the dataset, which decreases the processing time.

Motivation. Security-critical data is collected from a variety of sources within an enterprise to detect cyber attacks. However, not all of the collected data contributes to the detection process. For example, a network sniffer captures zero-byte data that is not useful in detecting cyber attacks as zero-byte flows are primarily used for handshaking in a TCP/IP connection [96]. Therefore, such useless data should be removed from the rest of the data captured by a network sniffer.

Description. Figure 2.11 shows the various components of Unnecessary Data Removal tactic. *Data collector* component collects security event data from various sources within an enterprise. The *data storage* component stores the collected data. Before forwarding data for analysis, data is intercepted by the *data cleaning* component, which filters the data and removes unnecessary data from the dataset. After the unnecessary data is removed, the rest of the data is forwarded to the *data processing* component that analyses the data to detect cyber attacks. Finally, the analysis results are visualized through the *visualization* component.

Constraints. This tactic requires that the data cleaning functionality is not computationally expensive, otherwise, it would cancel the benefit of quick response acquired through data size reduction. The rules for filtering out unnecessary data need to be designed carefully to ensure that data that is critical for accurately detecting cyber attacks does not get filtered out. These data filtration rules vary from situation to situation as data that is of significant value in one situation may not be significant in another situation [S67].

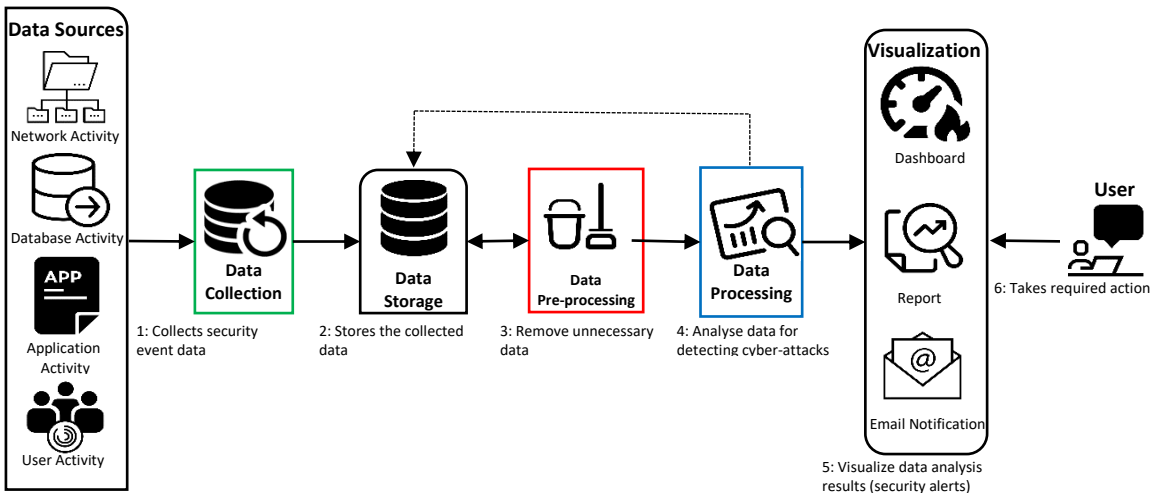


Figure 2. 11 Unnecessary data removal tactic

Example. The following two systems incorporate Unnecessary Data Removal tactic.

- Dynamic Time Threshold [S58] analyses security event data collected from various sources that include web access log, network log, host log, and network behaviour log. This system incorporates Unnecessary Data Removal tactic to remove log instance with URL suffix JPG, GIF, JPEG, WAV, CSS and JS. These log records automatically get stored in the web access log when a user requests a web page related to JS, CSS, or image data. However, these log records do not contribute to the threat analysis process, therefore, these are considered unnecessary and are removed.
- Batch and Stream Analyzer [S66] collects and analyse NetFlow data for detecting cyber attacks. This system employs Unnecessary Data Removal tactic for removal of zero-byte flows captured in the NetFlow data. As mentioned, zero-byte flows are used for handshaking in TCP/IP connection and have no relevance to threat detection, therefore, such data is removed from the rest of the NetFlow data.

Dependencies. Unnecessary Data Removal tactic requires Parallel Processing tactic (Section 2.5.1.4) to speed up the process of removing unnecessary data from the raw data collected from different sources.

Variations: Removal of duplicates. This tactic as described removes data that does not contribute to attack detection process. However, the collected data also contains a lot of duplicate records, whose analysis puts an extra burden on the computational process without any valuable contribution to the detection process. Therefore, specific representative instances of such records should be included, and duplicates should be removed before forwarding data to the analysis component. The removal of duplicates not only supports fast processing but also gives a clearer view of the activities directed towards or within a network [S6]. Compression Model for IDS [S21] implements this tactic on training data for IDS. The system uses affinity propagation [97] to remove duplicates from the training data and so extract a small dataset from a large-scale dataset. It was found that the incorporation of removal of duplicate tactics along with horizontal compression enhances the efficiency by 184 times with less than 1% negative effect on the attack detection accuracy. Multistage Alert Correlator [S6] incorporates this tactic for removing duplicate alerts in an alert correlation system designed to correlate individual alerts to determine attack patterns that can be used for predicting future attacks.

2.5.1.3 Feature selection

Introduction. Feature Selection tactic has been found in [S9], [S10], [S12], [S13], [S14], [S15], [S31], [S32], [S33], [S55], [S57]. This tactic selects and extracts the most relevant features from the network traffic data that can be analyzed for detecting cyber attacks. The incorporation of this tactic helps reduce storage volume, increase data processing speed, and reduce data complexity. This tactic not only improves the response time of a system but also contributes to the detection accuracy improvement.

Motivation. According to a Cisco report, global IP traffic was 1.2 Zettabytes in 2016, which is expected to reach 3.3 Zettabytes in 2021 [98]. Each IP traffic record contains more than 40 features (such as source IP address, destination IP address, source port, and destination port). For instance, the traffic records contained in Knowledge Discovery in Databases cup 1999 (KDDcup99) dataset [99] consist of around 41 features while traffic records in Centre for Applied Internet Data Analysis (CAIDA) dataset [100] has around 50 features. Many enterprises do not have the high computational capability for such a large size of data (using all available features). It is important to carefully select and extract specific features from the captured network traffic for analysis.

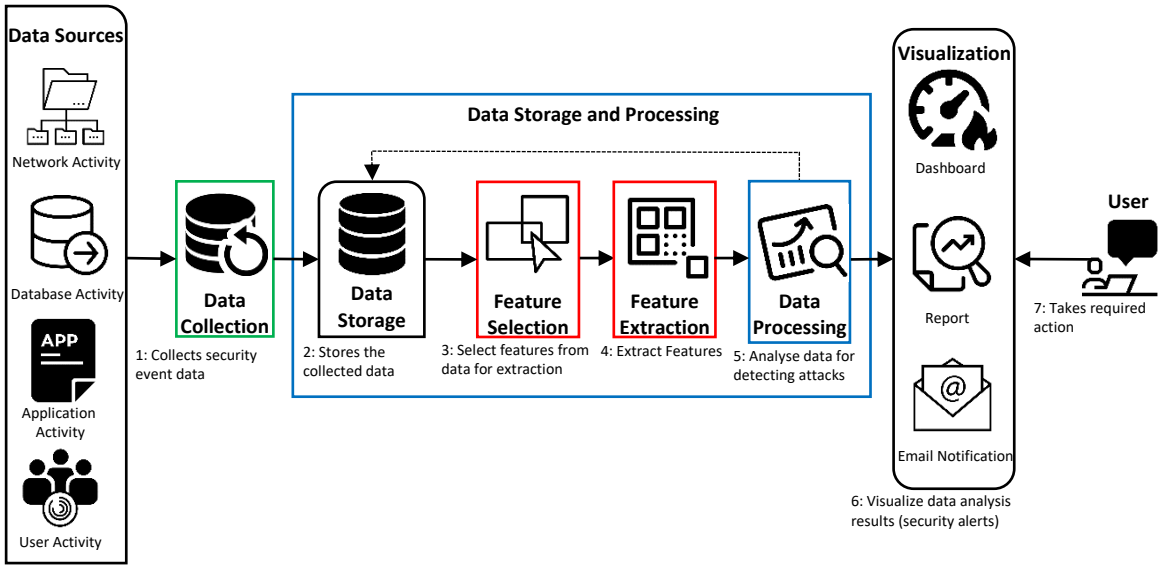


Figure 2. 12 Feature selection tactic

Description. Figure 2.12 shows the major components of Feature Selection tactic. The *data collection* component collects network traffic data from various sources (e.g., switch, router, or firewall). Each record in the captured data contains more than 40 features from which the specific features for each IP traffic record using the feature selection component. The *feature extraction* component extracts the specific features. The extracted feature dataset is passed onto the data processing component that helps detect cyber attacks, that lead to the generation of alerts presented to users through the *visualization* component. Once such attack alerts come under notice, a user or enterprise can take necessary steps to prevent or mitigate the effects of the attack.

Constraints. The Feature Selection tactic facilitates real-time security analytics. There must be a careful approach to ensure that any feature that contributes to the detection process should not be missed, otherwise, a system's accuracy would be severely affected. This condition becomes more relevant in cases where the network traffic is constantly evolving due to emerging technologies and new attacks. The feature selection and extraction operations should not be computationally heavy, otherwise, it would cancel the benefit of the reduced response time achieved through the reduction in data size.

Example. The following two implementations demonstrate the Feature Selection tactic.

- Cloud-based Threat Detector [S10] collects and analyze various types of data (e.g., system logs, firewall logs, and router logs) to detect cyber attacks such as DDoS attack and port scanning attack. This system leverages Naïve Bayes algorithm to select five features out of 50 features in CAIDA dataset. These features include source IP address, destination IP address, port, packet length, and protocol. After selecting the features, parallel processing framework (i.e., MapReduce) is used to extract the features from the records. The incorporation of Feature Selection tactic reduces the size of dataset from 200 GB to 50 GB. It was found that the incorporation of this tactic does not have any negative impact on the detection accuracy.
- Quasi Real-Time IDS [S12] leverages Information Gain Ranking Algorithm to select eight features from network traffic records in CAIDA dataset. After selecting the features, parallel processing framework (i.e., MapReduce) is used to extract the features from the records. A distinguishing characteristic of this system is that it also allows user to select features manually at runtime. This flexibility is helpful in situations where the dynamics of network traffic changes frequently. Such manual selection of features is achieved through the incorporation

of Tshark [101] and Apache Hive[8]. Tshark enables a user to select the features and Hive provides the table for storing the features.

Dependencies. Feature Selection tactic requires Parallel Processing tactic (Section 2.5.1.4) to speed up the process of feature selection. It is worth noting that Unnecessary Data Removal tactic (Section 2.5.1.2) and Feature Selection tactic are not interchangeable. Both tactics play separate roles in improving the response time of the BDCA system.

2.5.1.4 Parallel processing

Introduction. Parallel Processing tactic can be found in all (i.e., 74) of the reviewed studies. This tactic distributes the processing of a large amount of security event data among different nodes of a computing cluster. The nodes process the data in parallel fashion, which significantly improves the response time of a system.

Motivation. There are a number of sources within an enterprise that generate security event data. These sources include but not limited to network devices (e.g., switches and routers), database activities, application data, and user activities. Security event data is generated at a very high speed. For example, an enterprise as large as HP used to generate around one trillion security events per day in 2013, which was expected to grow further in the future years [15]. A standalone computer that processes such a large size of security event data in a sequential manner will take a lot of time to detect an attack, which is not tolerable in such security-critical situations.

Description. Figure 2.13 shows the main components of Parallel Processing tactic. The numbers in the figure show the sequence of operations. The *data collection* component collects security event data from different sources depending on the type of security analytics and security requirements of an enterprise. The *data collection* forwards the collected data to *data storage* component, which stores the data. Data can be stored in several ways such as Hadoop Distributed File System (HDFS), HBase, and Relational Database Management System (RDBMS). In order to enable parallel processing, the stored data needs to be partitioned into fixed-size blocks (e.g., 64MB or 128MB). After partitioning, data is processed by the *data processing* component through several nodes working in parallel according to the

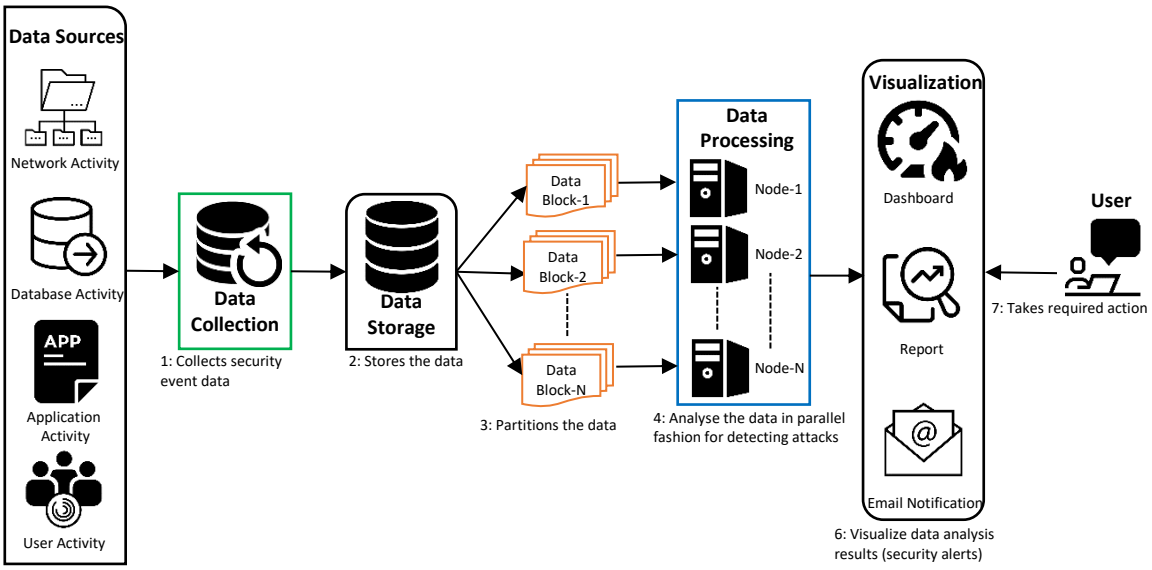


Figure 2. 13 Parallel processing tactic

[8] https://hive.apache.org/

guidelines of a distributed framework such as Hadoop or Spark. The result of analysis is shared with users through the *visualization* component.

Constraints. Parallel Processing tactic assumes that a BDCA system incorporating this tactic is already integrated with a cluster of nodes capable of processing data in a parallel fashion. Another important factor that needs to be taken care of is the breaking of a logical record across two blocks during the partitioning of data into blocks. It is important to keep enough information about the file data type so that record can be reconstructed.

Example. Honeypot-based Phishing Detection [S17] demonstrates the applicability of this tactic for improving the response time of a phishing attack detection system. The authors compared the response time of a sequentially implemented system with a parallel-implemented system, each processing 268 GB of security event data. It was found that sequentially implemented system took 180 minutes to process all data. On the other hand, parallel implementation of a system with Hadoop and Spark frameworks took 21 minutes and 14 minutes respectively with a cluster of 9 nodes. The authors also demonstrated that the more the number of nodes in a parallel processing scenario, the faster will be the response time of a system. For example, the response time of Hadoop was recorded as 57, 36, and 21 minutes with 3, 5, and 9 nodes respectively.

Dependencies. Parallel Processing tactic depends upon the Dynamic Load Balancing tactic (Section 2.5.3.1) and Data Ingestion Monitoring tactic (Section 2.5.4.1) for balancing the load among the nodes and controlling the flow of data into the nodes respectively.

2.5.1.5 Result polling and optimized notification

Introduction. Result Polling and Optimized Notification tactic is found in Count Me In [S56]. This tactic helps optimize the delay caused due to a predefined time interval for feeding the results from the mapper nodes into the reducer nodes inside a parallel processing BDCA system. This tactic ensures that as soon as values inside the mapper nodes change to a sufficient degree (set by an admin), the mapper node notifies the reducer node and accordingly forward the updated results to the reducer.

Motivation. MapReduce is a parallel processing framework that is widely adopted in a distributed setup [102]. This framework consists of two phases – (1) Map and (2) Reduce. The Map phase maps the features derived from the network traffic to a key and a value through multiple mapper nodes. For example, failed connections between a source 's' and a destination 'd' can be presented in the form of a key-value pair (s, d). In the Reduce phase, the key-value pairs generated by mapper nodes are fed into multiple reducer nodes for generating results. The generated results are evaluated against a predefined threshold through a trigger and if the results exceed a specific limit, then an alert is generated that signal towards a possible cyber attack.

The key-value pairs from mapper nodes are fed into reducer nodes after a predefined time interval (e.g., 5-minute aggregation or 1-hour aggregation). Once reducer generates the results, the trigger executes the predefined threshold. This predefined time interval introduces a delay. For example, an attack may be launched at t = 5 sec of the predefined time interval and the trigger will be executed at t = 5 min. In this 5 min, it is quite possible that a significant damage might already have been caused.

Description. The incorporation of Result Polling and Optimized Notification tactic in a BDCA system is demonstrated in Figure 2.14. *Data collection* component collects security event data from multiple sources. The collected data is stored by the *data storage* component. Next, *parallel data processing* component reads the stored data. The *parallel data processing* component consists of mappers and

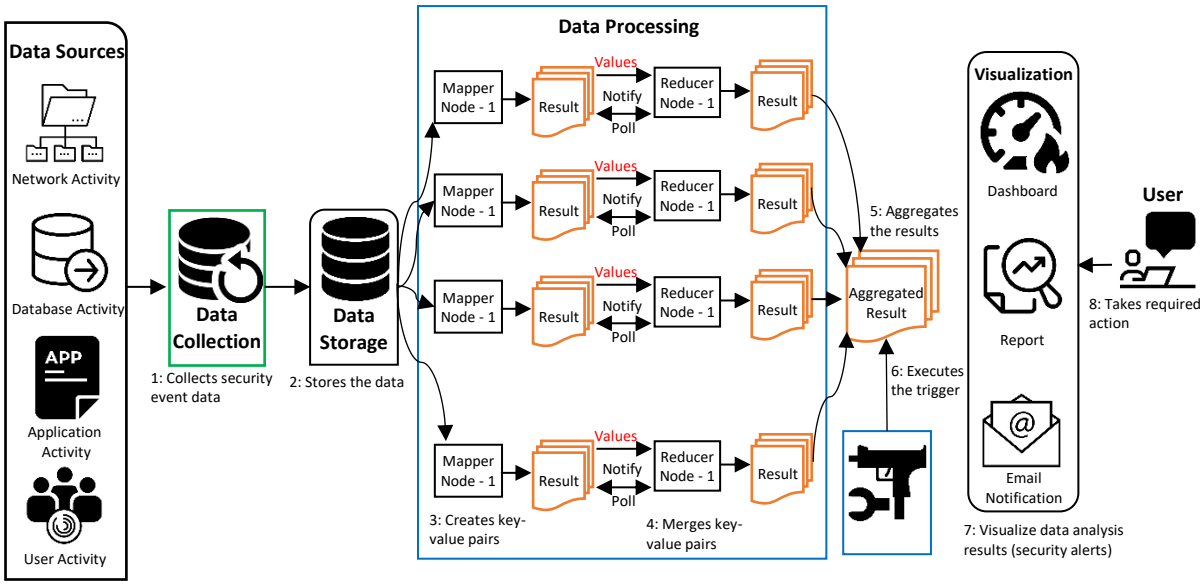


Figure 2. 14 Result polling and optimized notification tactic

reducers. The number of mappers and reducers depends upon the number of nodes in an underlying cluster. The mapper sub-components (e.g., Mapper Node-1 and Mapper Node-2) read the data in parallel and produce intermediate key-value pairs shown as result in Figure 2.14. When all the map jobs are completed, the key-value pairs (shown as values in the figure) are passed on to the reducers, which merge the values to produce the final results. Typically, the reducers have to wait until all map jobs are completed, which causes a significant delay in responding to cyber attacks.

To address this issue, the mechanism of the poll and notify is introduced. According to this mechanism, the mappers monitor the changes in incoming security event data. As soon as the change in the security event data crosses a predefined threshold, the mappers notify the reducers to get ready for taking the required data without waiting for completion of the predefined time interval. In addition to the optimized notification from mappers, the reducers can poll the mappers for results depending upon the processing capacity of the reducer(s). After receiving the intermediate key-value pairs, the reducers process the intermediate results to generate the final results. Immediately after the reducers produce the final results, the trigger is executed to check for possible cyber attacks.

Constraints. In a distributed setup, it is important that the threshold set for change in intermediate values at mapper nodes is applied to the average of all mapper nodes. For example, if a change in values at a single mapper node crosses the defined threshold, but the rest of the mapper nodes are not experiencing sufficient change in values, intermediate results should not be passed to the reducer nodes. This is due to the reason that by-passing the defined time interval and forwarding unchanged intermediate results from mappers to reducers will put extra processing burden on reducers without any significant gain.

Example. Count Me In [S56] experimentally examines the communication overhead caused by the incorporation of this tactic. The system monitors an uplink traffic averaging 1 Gb/sec during day-time hours. It is observed that out of the total messages communicated among the nodes of a system, only 0.40% relates to Result Polling and Optimized Notification tactic. In total out of 69, 810 notifications from mapper node to reducer nodes, the reducer nodes ignore 27, 704 notifications to limit simultaneous outstanding key updates.

Dependencies. Result Polling and Optimized Notification tactic requires Parallel Processing tactic (Section 2.5.1.4) to enable the reducer nodes to share the extra load generated due to result polling

among the nodes. This tactic also depends upon the MapReduce tactic (Section 2.5.3.2) for providing the programming framework to design mappers and reducers.

Variations: User-guided poll and notify. Result Polling and Optimized Notification tactic requires either the mappers to notify the reducers about updates or requires reducers to poll the mappers for updates. However, in some systems, the task of poll and notify can be relegated to a user. A user can initiate the process of sending intermediate key-value pairs from mappers to reducers at any point of time irrespective of the predefined time interval. For example, SEAS-MR [S38] has both the options – periodic aggregation and user-guided aggregation of intermediate results produced by mappers. The user-guided poll and notify can also be used for long-term analysis where a user can specify the start time and the end time for an interval. In this system, periodic aggregation is implemented with MapReduce as it is better for performance while user-guide aggregation is implemented with Pig[9] script as it facilitates user interactivity.

2.5.1.6 Data cut-off

Introduction. Data Cut-off tactic has been identified from Forensic Analyzer [S30] and VALKYRIE [S45]. This tactic applies a customizable cut-off limit on each network connection or process to select and store data pertaining to a specific portion of the connection or process. For example, selecting and storing only first 15 KB of network traffic data for a connection or data pertaining to first 100 sec of the execution time of a process. Such a cut-off reduces size of the dataset for security analysis, which helps improve the overall performance of the system.

Motivation. Due to the ever-increasing volume of security relevant data (e.g., network traffic, system logs, and application activity), it is infeasible to collect, store, and analyse the data in its entirety. For example, Lawrence Berkeley National Laboratory (LBNL), a security research lab containing around 10,000 hosts, experiences around 1.5 TB of network traffic per day. In majority of the cases, only a small subset of the security event data turns out to be relevant for security analysis [103]. In case of network connections, more connections are short with few large connections accounting for bulk of total volume. Thus, by selecting and storing the first N bytes (cut-off) for each large connection, which can be stored in its entirety. Such connections' beginning portions contain the relevant information such as protocol handshakes, authentication logs, and data item names.

Description. Figure 2.15 shows the main elements of Data Cut-off tactic with the numbers to indicate the sequence of the operations. The *data collection* component collects security event data from one or several available sources and passes the collected data to *data cut-Off* component. Examples of security events include network security event (e.g., [source IP, destination IP, port, protocol]) and process event (e.g., [file name information, privilege level, parent process ID, timestamp]). The *data cut-Off* component enforces a cut-off by discarding security events that appear after a network connection or process has reached its predefined limit. Any security event that appears after the predefined limit does not contribute significantly to the attack detection process, therefore, analysing such securityevents put an extra burden on data processing resources without any significant gain. The cut-off limit (i.e., first N of a connection or process) is determined by the security operators based on their experience and operating variables. For example, a higher cut-off limit (e.g., 40% of a process) is applied in non-working hours due to the lower speed of data generation as compared to a lower limit (e.g., first 15% of a process) during working hours. The security event data left after cut-off is stored by the *data*

[9] https://pig.apache.org/

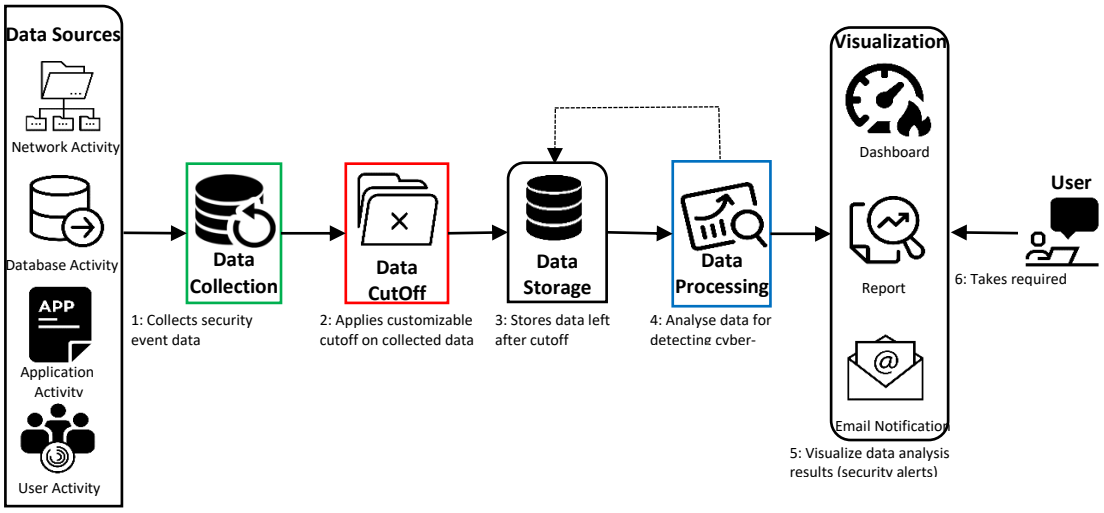


Figure 2. 15 Data cut-off tactic

storage component. The stored data is read by *data processing* component to analyse it for detecting cyber attacks. Finally, the result of analysis is displayed to a user through *visualization* component. A user takes the required action upon arrival of any outstanding alerts.

Constraints. Data Cut-off tactic poses a risk of evading an attack from detection. If an attacker is smart enough to initiate an attack after the cut-off limit, a BDCA system will not record the malicious activity, and so the attack will go undetected. There are several delaying tactics usually employed by attackers with the intent to evade such tactics that record only the initial portion of an interaction. Several compensation techniques can be employed to reduce the risk of such an evasion. These techniques include: (1) selecting cut-off limit according to the type of data sources as delaying an attack to later stages is harder for some services as compared to others (2) the cut-off limit can be increased for data sources where there is a higher risk of an attack (3) the application of cut-off can be randomized instead of always being applied at the start of an interaction (e.g., network connection or a process) so that an attacker cannot predict at which point the cut-off will come into play. In addition, the functionality required for implementing Data Cut-Off tactic should not be computationally expensive, otherwise, it will cancel the benefit that is expected through the incorporation of this tactic.

Example. Data Cut-Off tactic has been implemented in Forensic Analyzer [S30] and VALKYRIE [S45].

- Forensic Analyzer [S30] is a cloud-based network security forensic analysis system that is evaluated by detecting phishing attacks based on analysis of captured network traffic. The data collected in a span of six months is around 20 TB in size. Data Cut-Off tactic was applied to the collected data with a cut-off limit of 15 KB. This means to only select the first 15 KB of each network connection. The incorporation of this tactic reduced the size of data from 20 TB to 1 TB. The effect of data cut-off on detection accuracy of a system has not been reported.
- VALKYRIE [S45] is a BDCA system that detects malware attacks based on an analysis of kernel-level telemetry data. The system implements Data Cut-Off tactic to reduce the size of data by selecting the data pertaining to only first 100 seconds of the execution time of a process. Despite the incorporation of Data Cut-Off tactic, the system achieves detection accuracy of around 97-99%.

Dependencies. Data Cut-Off tactic requires Parallel Processing tactic (Section 2.5.1.4) to speed up the process of discarding security event data that appears after the predefined cut-off limit.

2.5.2 Accuracy

This section reports the architectural tactics that are related to Accuracy quality attribute.

2.5.2.1 Alert Correlation

Introduction. Alert Correlation tactic have been found in [S6], [S7], [S11], [S16], [S19], [S38], [S39], [S41] and [S65]. This tactic analyses the individual alerts produced by security system(s), discards the irrelevant alerts, and groups together the relevant alerts based on a logical relationship between them to provide a global and condensed view of the security situation of an organization's infrastructure (i.e., network and hosts). The incorporation of this tactic improves the accuracy of a BDCA system by reducing the number of false positives and detecting highly sophisticated and complex attacks.

Motivation. Organizations employ different security technologies for better detection coverage of their networks and hosts. For example, a typical organization may deploy firewall, anti-virus, and IDS for accepting/dropping network traffic, scanning malware based on predefined signature, and detecting known attack patterns or abnormal behaviours respectively. Unfortunately, these security tools generate a large number of alerts. For instance, an IDS deployed in a real-world network generates around 9 million alerts per day [104]. Investigating and responding to these many alerts is quite challenging especially when 99% of them are false positives [105]. Furthermore, these security tools monitor the security in isolation without considering the context and logical relationship between alerts generated by other security tools. This isolated approach is not able to detect attacks that operate in slow mode over a period of time where some alerts are precursors to more complex and dangerous attacks. To address these issues, a high-level management is required that correlate the alerts by taking context and their logical relationship into consideration before reporting them to users.

Description. Figure 2.16 shows the main components of implementing Alert Correlation tactic. The *data collection* component collects security event data from different sources. The collected data is stored in the *data storage* and copied to the *data pre-processor* component for pre-processing the raw data. The pre-processed data is ingested into the *alert analysis* component, which analyses the data for detecting attacks. It is worth mentioning that Alert analysis component analyses the data in an isolated fashion (without considering any contextual information) either using misuse-based analysis or anomaly-based or both. The generated alerts are transferred to the *alert verification* component, which uses

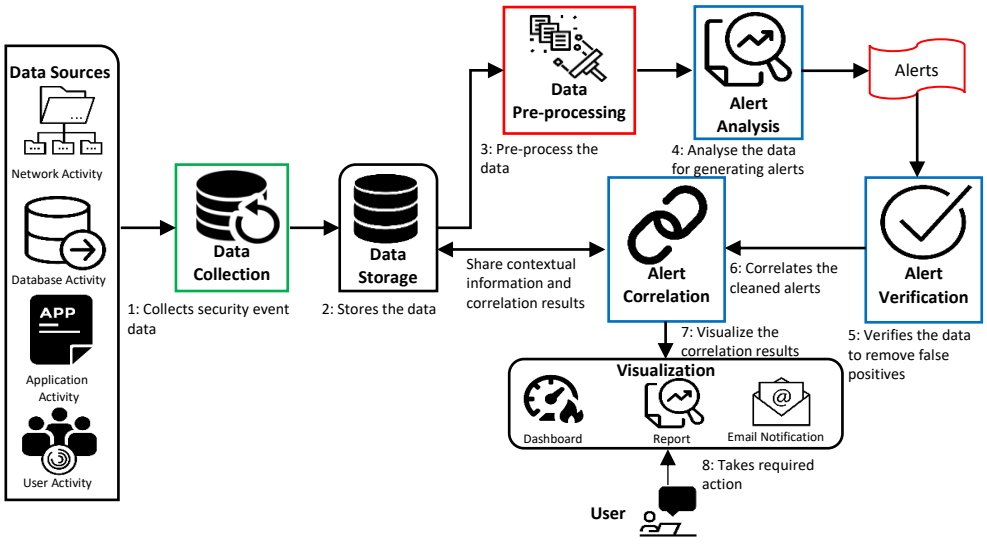


Figure 2. 16 Alert correlation tactic

different techniques [106] to determine whether an alert is a false positive. The alerts identified as false positives are discarded at this stage. The clean and synthesized alerts are forwarded to the *alert correlation* component. The alerts are correlated (i.e., logically linked) using different techniques [106] such as scenario-based correlation, rule-based correlation, statistical correlation, and temporal correlation. The Alert correlation component coordinates with *data storage* for taking the required contextual information about alerts. The results of correlation are released through the *visualization* component. Finally, either an automated response is generated, or a security administrator analyses the threat and responds accordingly.

Constraints. The in-depth analysis employed for alert correlation improves the accuracy but increases the response time due to the inclusion of an extra complex computation stage to a system. The tactic also requires mechanisms for acquisition of domain knowledge and adaptation to changes in the networks and host infrastructure [106].

Examples. This review has identified nine systems that implement Alert Correlation tactic. These systems collect individual alerts from different security technologies (e.g., IDS or Firewall) for detecting attacks. Each of them applies different correlation techniques, some of which have been described below.

- GSLAC [S7]: The system employs causal-based technique for correlating alerts. Each alert is treated as a vector with multiple attributes (i.e., destination IP, source IP, timestamp and so on). The alerts are presented in the form of a graph where each alert has a prerequisite alert and a consequence. A security analyst analyses the graph for identifying complex attack scenarios.
- Hunting attacks in the dark [S41]: This system correlates alerts based on their source and destination IP addresses. The similarity between IP addresses for different alerts is measured using intersection cardinalities and if the similarity score is above a predefined threshold, it means alerts belong to same IPs that signals towards a potential attack.
- Multistage alert correlator [S6]: This system employs a model of prerequisites and consequences proposed in [107] for determining the relationship among individual alerts. The alerts are correlated based on similarities among their source IP, destination IP, start time, and end time. A graph is generated that shows each alert with its prerequisite and consequence. If a prerequisite of an alert is present in the graph as a consequence of a previous alert, the two alerts are closely related and analysed for picturizing the complex attack scenario.

Dependencies. Alert Correlation tactic will correlate alerts of any quality; however, effective correlation requires alerts to be of a good quality that is dependent upon the tactics employed in the data processing component such as Attack Algorithm Selection tactic (Section 2.5.2.3) and Signature-based Detection (Section 2.5.2.2).

2.5.2.2 Signature-based detection

Introduction. Genetic Algo-based Distributed Denial of Service (DDoD) Detection [S61], Hybrid Intrusion Detection [S72], and Snort + PHAD + NETAD [108] employ this tactic. In addition to the anomaly detector that detects attacks based on deviation from the learned normal behaviour, the Signature-based Detection tactic analyse the collected security event data for two objectives to find a match with the already available attack patterns or signatures. In either case, finding a match or a

deviation, a system generates an alert signalling a possible cyber attack. The combination of misuse and anomaly significantly improves the detection accuracy and reduces the false positive rate.

Motivation. Based on detection principle, BDCA systems are of two types – Signature-based (often called misuse-based) and Anomaly-based. Signature-based systems detect attacks based on predefined attack patterns. These patterns are designed based on already reported attacks. If an ongoing activity matches with attack patterns, the activity is termed as malicious. These types of systems are very effective in detecting known attacks but are unable to detect unknown attacks [109]. Anomaly-based systems learn the normal behaviour of an organization's infrastructure and any activity that deviates from this behaviour is termed an attack. This class of systems can detect unknown attacks, however, it generates a large number of false positive alarms [110]. Considering the limitations of both type of systems, it is important to come up with a solution that can minimize these limitations.

Description. The main components of Signature-based Detection tactic are shown in Figure 2.17. The *data collection* component collects security relevant data from different sources. The collected data is stored by the *data storage* component. Next, data is fed into the *signature-based detection* component that analyses the data to identify attack patterns. For such analysis, this component leverages the pre-designed rules from the *rules database* that define attack patterns. If a match is identified, an alert is directly generated through a *visualization* component. If *signature-based detection* component does not detect any attack pattern in the data, the data is forwarded to the *anomaly-based detection* component for detecting unknown attacks that cannot be detected by the *signature-based detection* component. The *anomaly-based detection* component analyses the data using ML algorithms to detect deviations from the normal behaviour. When an anomaly (deviation) is detected, an alert is generated through the *visualization* component. At the same time, the *anomaly* is defined in the form of a *rule* or attack pattern and added to the *rules database*. This way the *rules database* is constantly updated to enable the *signature-based detection* component to detect a variety of attacks.

Constraints. Integrating signature-based and anomaly-based attack detection techniques may impact the overall response time of a system by introducing additional data analytics requirements. Furthermore, combining signature-based and anomaly-based detections into a single BDCA system and getting them interoperate in an efficient and successful way can be more complex and challenging.

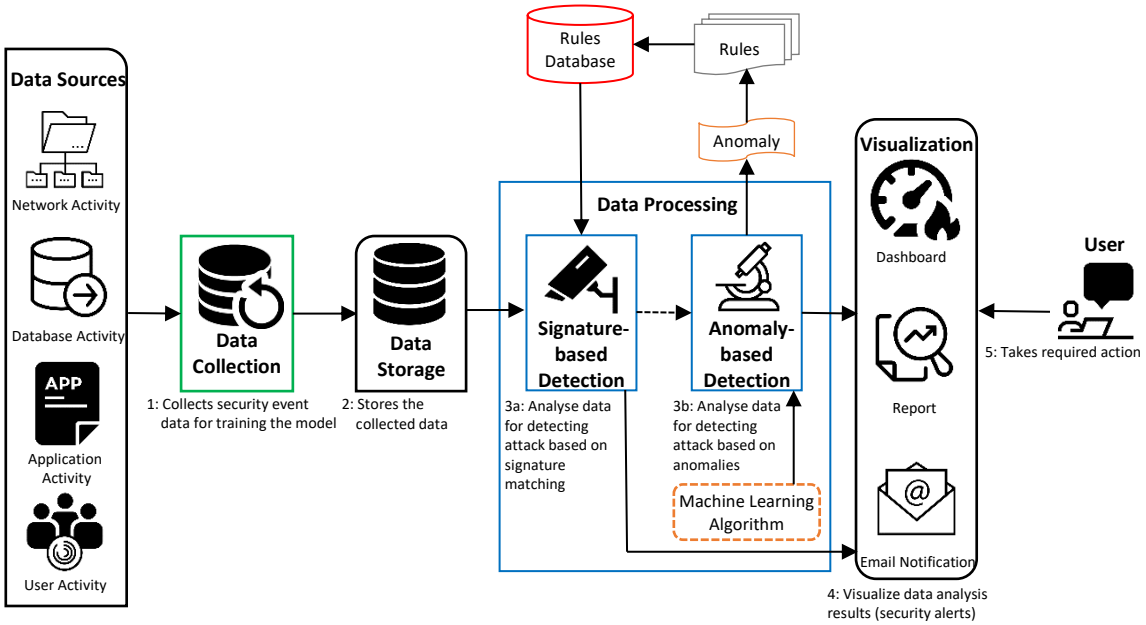


Figure 2. 17 Signature-based detection tactic

Example. The following two systems demonstrates the implementation of Signature-based Detection tactic.

- Hybrid Intrusion Detection [S72]: This system combines signature-based IDS (Snort) with an anomaly-based detector implemented using Hadoop and Hive. The hybrid system is compared with the stand-alone signature-based system in terms of accuracy. Both the systems have been evaluated with several attacks such as ICMP, Smurf, SYN flood, UPD, and Port scanning attack. It is found that a hybrid system achieves higher detection rate for all attacks compared to a stand-alone system. For example, for port scanning attack, a stand-alone system achieves a detection rate of around 40% while a hybrid system shows a detection rate of around 50%.
- Snort + PHAD + NETAD [108]: This system combines signature-based IDS (Snort) with two anomaly detectors among which one (PHAD) detect anomalies based on packet header analysis and other (NETAD) detects anomalies based on network traffic analysis. The hybrid system is evaluated using IDEVAL dataset [56] to explore its detection rate in comparison to the stand-alone signature-based system. It is observed that the stand-alone system is able to detect only 27 attacks while the hybrid system (i.e., Snort+PHAD+NETAD) detects 146 attacks from the dataset.

Dependencies. Signature-based Detection tactic requires Parallel Processing tactic (Section 2.5.1.4) to reduce the additional analysis time required due to two-phase analysis (i.e., signature-based and anomaly-based). Additionally, this tactic also depends upon Attack Detection Algorithm Selection tactic (Section 2.5.2.3) to efficiently select an effective algorithm for detecting anomalies in the security event data.

2.5.2.3 Attack detection algorithm selection tactic

Introduction. We have already stated that all of the reviewed systems leverage some type of ML algorithm to analyse the security event data for detecting cyber attacks, therefore, this tactic can be found in almost all of the reviewed systems. Unlike ML Algorithm Optimization tactic (Section 2.5.1.1) that emphasizes the role of ML algorithm with respect to response time, the objective of Attack Detection Algorithm Selection tactic is to highlight the role of suitable algorithms in improving the accuracy of a BDCA system and provide some guidelines for selecting ML algorithms that are most appropriate for accurate detection of attacks.

Motivation. BDCA systems are expected to accurately detect cyber attacks without generating security alerts for legitimate activities (i.e., false positives). In addition to several other factors such as data source and data quality, the employed ML algorithms play a significant role in accurately detecting cyber attacks. BDCA systems use ML algorithms to classify the security event data either as legitimate (corresponding to normal activity) or malicious (corresponding to attacks). There exists a variety of ML algorithms that can be leveraged to detect attacks. These algorithms include but not limited to Logistic Regression, Naïve Bayes, Support Vector Machine, Random Forest, and Gradient Boosted Decision Tree, K-means, and Artificial Neural Networks. For details on ML algorithms used in cyber security analytics, readers should refer to [59]. The selection of ML algorithm is challenging because an algorithm working best in one scenario or with data may not work as efficiently in another scenario or with another data. For instance, [39] shows that logistic regression is the most accurate ML algorithm for KDD dataset while random forest is the most accurate for NSL-KDD dataset. Similarly, another study [41] on BDCA reveals that the accuracy of conjunctive rule dropped from 80.1% to 78.95% as the

training data size is reduced from 100% to 10%. Hence, the big challenge here is to determine which of the many available algorithms will yield the most accurate results.

Description. The major components of Attack Detection Algorithm Selection tactic are shown in Figure 2.18. The *data collection* component collects security event data for training the BDCA system for detecting cyber attacks. The training data can be collected from sources within an enterprise where a system is supposed to be deployed as depicted in Figure 2.18 or an already available popular dataset such as KDD can be used as training dataset. How much data should be sufficient to train the model varies from one case to another. For example, the famous DARPA 1998 dataset contains data collected from network and operating system in a span of nine weeks. The first seven-weeks data were assigned as training data and the last two weeks as testing data. After collecting the training data, the *data preparation* component prepares the data for training the model by applying filters and feature extraction techniques. Next, the prepared training data start training the attack detection model. Once the model is trained, it is tested to investigate whether the model can detect cyber attacks. For testing the model, the data is collected from an enterprise as shown in step 4. The test data is prepared for feeding into the attack detection model. The prepared test data is forwarded to the *attack detection model*, which analyses the data based on the rules learned during the training phase. Here, the incoming test data instances are classified as either legitimate or malicious. The analysis results are displayed to a user through the *visualization* component. In case of malicious or attack situation, a user can take immediate actions that may include blocking certain ports or cutting off the affected components from the network to stop further damage.

Constraints. While selecting an attack detection algorithm, software architect should keep an eye on several things due to the following reasons

- The selection of ML algorithm is tricky because an algorithm may work well for detecting one type of attack but may not work well for detecting other types of attacks
- Similarly, algorithm selection is challenging in the sense that in addition to accuracy, it affects other system's qualities such as response time, complexity, and understandability of the results. For example, Cheng et al. [95] compare SVM with Extreme Learning Machine (ELM) in terms of accuracy and performance. It has been revealed that SVM generates more accurate results but is computationally expensive. On the other hand, ELM generates less accurate

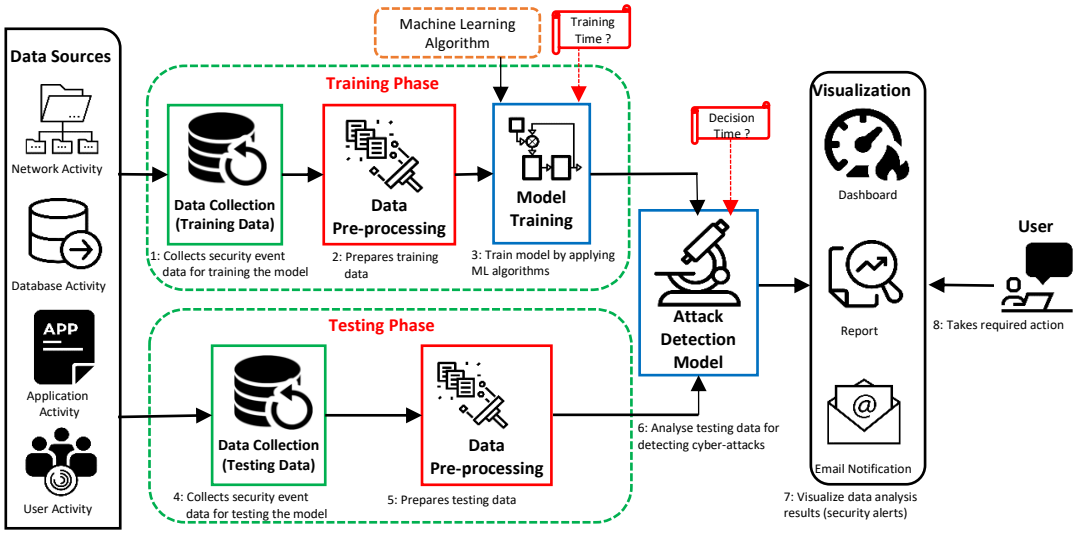


Figure 2. 18 Attack detection algorithm selection tactic

- results, but is more light-weight. A reasonable trade-off should be established among various system's qualities while selecting an algorithm.
- There is no such standard available that compares the accuracy of all ML algorithms on the same dataset. Different research explorations use different datasets (e.g., KDD and DARPA) or different subsets from the same dataset, which makes it harder to fully rely on these findings for selecting an algorithm for a specific BDCA system.
- The applicability of ML algorithms closely relates to the quality of the data, whose quality ought to be carefully assessed for selecting a suitable attack detection algorithm. For example, certain ML algorithms works well even with a smaller sample data while others require millions of samples for training. Similarly, some ML algorithms work well with numerical data while others are more suitable for categorical data.
- We also recommend descriptive analysis (e.g., determining standard mean and standard deviation) and visualization via graphs and plots to help understand the data, which eventually helps in the selection of suitable algorithm
- Some ML algorithms need to be tuned via hyperparameter optimization in order to use them to the maximum of their ability. Random search, grid search, and Bayesian optimization are the common techniques for hyperparameter optimization of the ML algorithms.
- The selection of algorithm also depends upon the working mode of a BDCA system [59]. An algorithm that best fits for offline analysis may not be suitable for online analysis.
- Whilst there is no silver bullet for the selection of an ML algorithm, some common characteristics of the algorithms can be kept in view during the selection. For instance, linear regression is quite suitable for data with a large number of features, decision tree can be used in cases where the interpretability of the results is a concern, and neural networks are used in scenarios where a decent computational resource is available.

Example. Whist all of the reviewed systems leverage ML algorithms, we report only those systems that explore and compare the effects of multiple algorithms on the attack detection rate of a system. The objective is to exhibit the importance of algorithm selection for the detection rate of a BDCA system.

- Streaming-based Threat Detection [S1]: This system explores the applicability of K-means clustering algorithm and Fuzzy C-means clustering algorithm for accurately detecting cyber attacks. The system is evaluated to detect DDoS and flooding attacks based on analysing 260 GB of network traffic data gathered at Chicago Equinix data centre [100]. Experimental results show that detection rate for K-means is 91.8% with 1.8% false positive while for Fuzzy C-means the detection rate is 86.5% with 2.7% false positive.
- Improved K-means IDS [S27]: K-means algorithm is improved by specifying criteria for selecting initial centre of cluster which is random in traditional K-means algorithm. Both K-means and improved K-means are implemented in IDS for comparing detection accuracy. It has been found that improved K-means accurately detects 89% of the attacks with 2.4% false positive rate in KDD dataset. On the other hand, K-means shows an accuracy of 86% with 1.0% false positive rate.
- Spark-based IDS Framework [S9]: This framework compares five ML algorithms - Logistic regression, Support vector machine, Random forest, Naïve Bayes, and Gradient boosted decision tree. Two datasets i.e., KDD and NSL-KDD are used to evaluate the accuracy of the

system with each of the algorithms. With KDD, Logistic regression shows the best accuracy (i.e., 91%) and SVM shows the worst (i.e., 78%). With NSL-KDD, the best accuracy is achieved with Random Forest (i.e., 82.3%) while SVM delivered the worst accuracy rate of 37.8%.

Dependencies. Attack Detection Algorithm Selection tactic depends upon Unnecessary Data Removal tactic (Section 2.5.1.2) and Feature Selection tactic (Section 2.5.1.3) to help bring the collected data into a refined form. After the application of Unnecessary Data Removal and Feature Selection tactic, attack detection algorithm can be efficiently applied to the refined data to accurately detect cyber attacks. Attack Detection Algorithm Selection tactic needs to be incorporated alongside ML Algorithm Optimization tactic (Section 2.5.1.1) as these two tactics establish the trade-off between the effects of the ML algorithm on accuracy and response time.

2.5.2.4 Combining multiple detection methods

Introduction. Combining Multiple Detection Methods tactic has been used in Multiple Detection approaches [S46]. This tactic applies and integrates the results of multiple security analytic methods on security event data. The incorporation of multiple security analytic methods in a BDCA system reduces false positive rate and improves the attack detection accuracy.

Motivation. Most of the BDCA systems employ a single detection principle for detecting attacks. Examples of detection principles include traffic volume to a server crosses a predefined threshold, a large number of port access in a certain period of time, and IP address that generate traffic above a certain threshold. The disadvantage of using a single detection approach is that a system usually generates a large number of false positives [S46], which can be reduced by using multiple detection approaches for the same security threat data. Each detector accommodates a different detection principle. For example, SYN flood attack can be detected by monitoring or analysing the number of network flows in network traffic data while a DoS attack can be detected by monitoring the volume of network traffic between source and destination IPs.

Description. The main components of Combining Multiple Detection Methods tactic are shown in Figure 2.19. Security event data is collected from different sources. It is worth noting that the sources from where security event data can be collected is not limited to what is shown in Figure 2.19. The choice of data sources varies from organization to organization depending upon their specific security

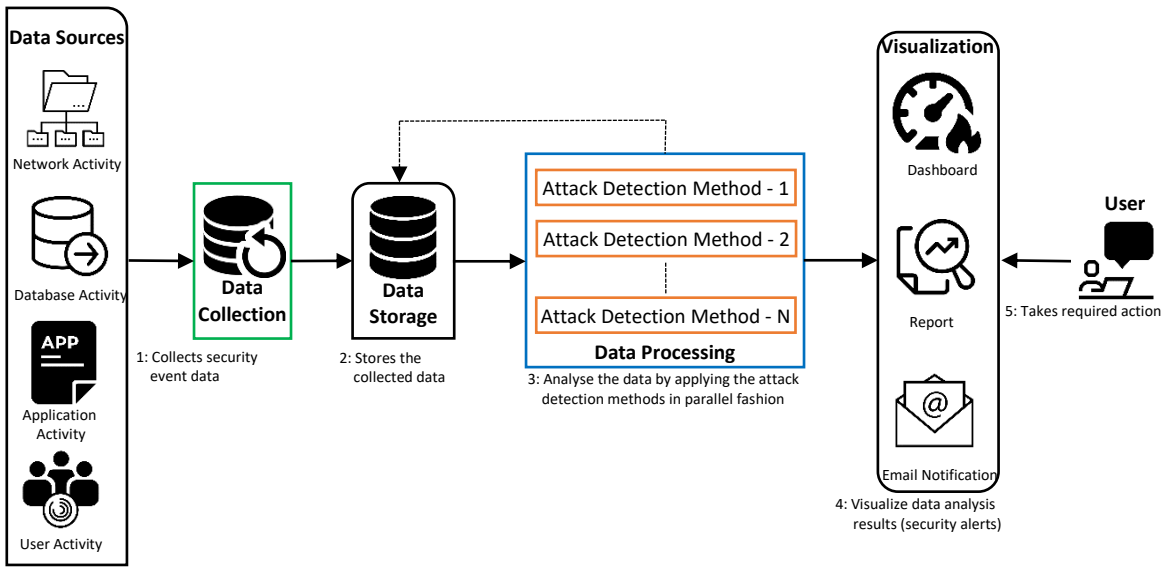


Figure 2. 19 Combining multiple detection methods tactic

requirements. After collection, data is stored in a *data storage*. Then the data is forwarded to the *data processing* module where various attack detection methods are applied to analyse the data. The number and choices of the attack detection approaches depend upon several factors, which include the processing capability of an organization, the data sources, security requirements, and an organization's security expertise. For example, a highly security-sensitive organization (e.g., National Security Agency) with high budget and computational power may incorporate a greater number of attack detection approaches to secure their data and infrastructure from cyber attacks. The attack detection approaches are applied to the whole dataset in a parallel fashion. The *visualization* component immediately reports any outstanding anomalies to users or administrators, who are expected to respond to security alerts.

Constraints. Combining Multiple Detection Methods tactic requires more computational power as compared to a BDCA system that incorporates a single detection approach. If a system uses multiple detection approaches, it will need more computation power; this requirement can affect the overall performance of the system.

Example. Combining Multiple Detection Methods tactic has been implemented in Multiple Detection approaches [S46]. This system integrates three detection approaches for analysing network traffic data. The first approach detects attacks based on finding anomalies with respect to the number of network flows. The second approach detects attacks based on finding anomalies with respect to the number of network packets. The third approach correlates the traffic volume with the IP flows to detect attacks. The system is tested with two types of attacks – DoS attack and SYN flooding attack. Under a DoS attack, the victim network experiences network flows that are large in size and small in number. On the other hand, under an SYN flooding attack, the victim network experiences network flows that are small in size but large in number. A BDCA system detects DoS attack through the second approach that detect anomalies based on number of network packets. On the other hand, SYN flooding attack is detected through the first approach that detects attacks based on number of network flows as the number of network flow during the attack situation crosses the pre-defined threshold.

Dependencies. Combining Multiple Detection Methods tactic depends upon Parallel Processing tactic (Section 2.5.1.4) that can provide the processing speed required for applying multiple attack detection methods on security event data. Without parallel processing, the system's response would be too slow.

2.5.3 Scalability

This section reports the architectural tactics that are related to Scalability quality attribute.

2.5.3.1 Dynamic load balancing

Introduction. Dynamic Load Balancing tactic can be found in GSLAC [S7] and Cloud Bursting [S34]. This tactic is used to balance the processing load among analysis nodes by dividing the security event data among the nodes. Having processing capacity available in a cluster, Dynamic Load Balancing tactic makes a system scale well without adding any further hardware resources.

Motivation. A BDCA system leverages a cluster of computing nodes for storage and analysis of security event data. The size of the cluster is different for different systems (e.g., 10 nodes in [S7] and 5 in [S34]). A system distributes the security event data among the nodes to speed up the process. When the speed of data input increases (for instance from 100 MB/sec in weekdays to 150 MB/sec over weekends), it is important that the system distributes the increased load in a balanced way to avoid a situation where

one node is under extreme load (i.e., 100% CPU utilization) and another node is under-loaded (i.e., 30% CPU utilization).

Description. Figure 2.20 shows the main components of Dynamic Load Balancing tactic. The *data collection* component collects data from different sources. The captured data is sent to the *data filtration* component, which removes the data that does not contribute to the attack detection process. The filtered data is forwarded to a *load balancer*, which distributes the data among different nodes to balance the workload among the nodes in the *data processing* component. Data can be distributed based on different criteria. For example, network traffic can be distributed based on header information (e.g., IP address or TCP ports) or payload information (e.g., signatures). The balanced distribution of data is not fully reliable due to the heterogeneous nature of the security event data [111, 112]. For example, if network traffic is distributed based on IP range, one range may contain a greater number of packets with large size, which may exhaust one node. On the other hand, a node handling another IP range with few packets of small size might be sitting idle. Therefore, the *resource monitor* component is introduced that constantly monitors the CPU utilization of nodes and reports to the *load balancer*. When the difference between the CPU utilization of the nodes crosses a predefined threshold, the *load balancer* rebalances the load among the nodes by moving processing load from overloaded nodes to less loaded nodes. After successful analysis of the data in the *data processing* component, the results are shared with users through the *visualization* component.

Constraints. This tactic assumes that a cluster has the processing capacity; otherwise, there is no chance of balancing the load if all nodes are already utilizing 100% of their CPU power. Furthermore, this tactic requires an efficient mechanism for selecting a target node and deciding the amount of data to be moved from the overloaded node to the target (under-loaded) node.

Examples. The following systems implement Dynamic Load Balancing tactic.

- GSLAC [S7]: During runtime, the resource monitor monitors the workload of the nodes and periodically updates the latency list that shows the CPU utilization of the nodes. When the difference between CPU utilization among the nodes crosses the threshold, the resource monitor alerts the load balancer. The load balancer uses dynamic hot spots migration [113] to rebalance the workload among the nodes.
- Cloud Bursting [S34]: This system leverages Dynamic Load Balancing tactic in a slightly different way. The resource monitor monitors the workload of the local cluster of nodes and

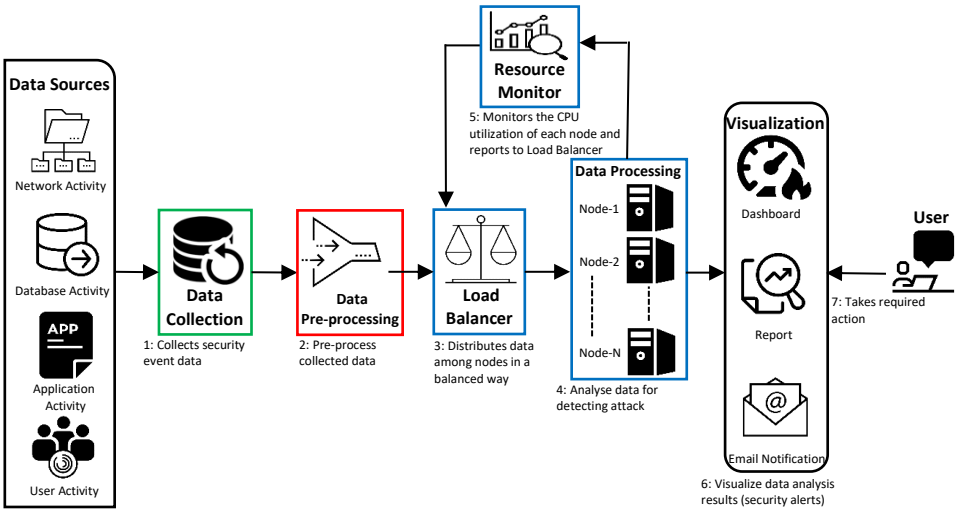


Figure 2. 20 Dynamic load balancing tactic

- keeps providing status information to a load balancer. Upon arrival of a new stream of security event data, the load balancer decides whether to launch the data processing job locally or burst it to other clusters on a cloud.

Dependencies. In principle, Dynamic Load Balancing tactic does not require any other tactic for its implementation. However, in a BDCA system, it will work in coordination with tactics like Unnecessary data removal (Section 2.5.1.2), Feature selection (Section 2.5.1.3), Data Cut-Off (Section 2.5.1.6), and Parallel Processing tactic (Section 2.5.1.4).

2.5.3.1 MapReduce

Introduction. This tactic has been found in all of the reviewed systems that use Hadoop platform for storing and analysing security event data. MapReduce tactic provides a programming framework for scaling software applications across a cluster of multiple nodes. Although MapReduce is a well-documented programming framework, this tactic captures the contribution of MapReduce relevant to the enhancement of scalability. The tactic abstracts various issues of scalability that include complex parallelization, synchronization, and communication mechanisms. In addition to security, MapReduce is also a widely accepted framework in other big data analytics domains such as bioinformatics [114, 115], astronomy [116, 117], and healthcare [118].

Motivation. In order to scale-out a BDCA system for handling a huge amount of data, a simple solution is to add more hardware resource to a system. The additional hardware can be added to the same physical machine (vertical scaling) or it can be added as a separate physical machine (horizontal scaling). It is quite challenging to efficiently handle complex parallelization, synchronization, communications, and resource utilization with the addition of hardware resources to an existing system. Therefore, a framework is required to handle these outstanding issues.

Description. The main components of MapReduce tactic are shown in Figure 2.21. The *data collection* component collects the security event data from one or multiple sources. The collected data is forwarded to *data filtration* component for removing data that does not contribute to attack detection. The filtered data is partitioned by the *master nodes* to store it in the HDFS files of *data nodes*. The *mapper* inside each data node reads its assigned HDFS block of the data for processing. It is important to note that the number of mappers does not depend upon the number of data nodes, rather, it depends upon

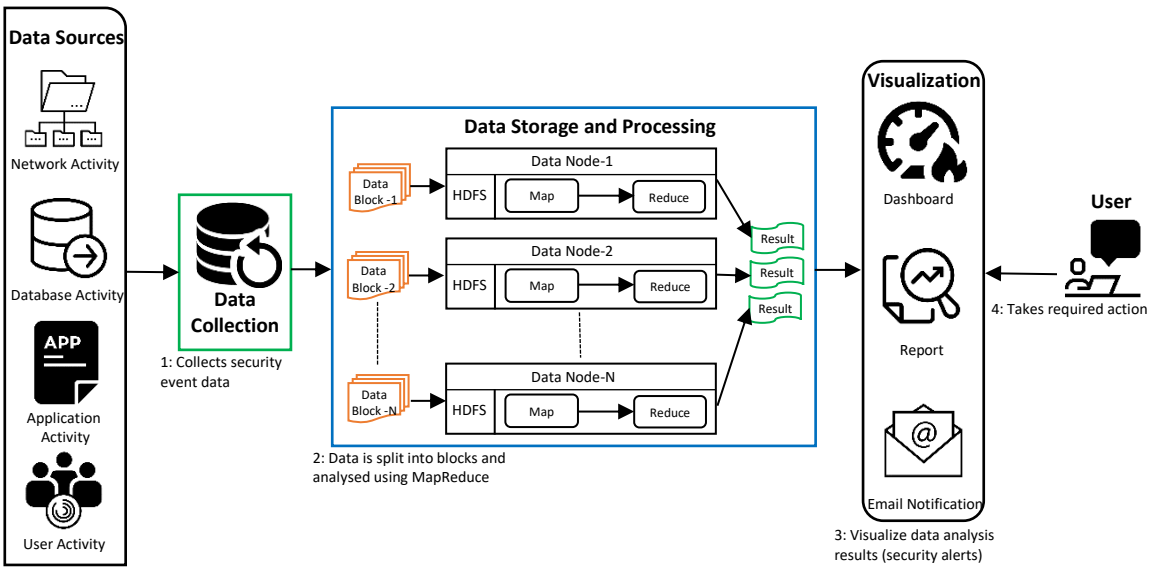


Figure 2. 21 MapReduce tactic

the number of input data blocks. The mappers process data simultaneously in the form of key-value pairs (key, value) to generate intermediate results (key, values list). The intermediate results are sorted, collated, and passed to the *reducers*, which merge and aggregate the intermediate results to generate the final output. The number of mappers and reducers running simultaneously depends upon the capacity, workload, and users' recommendations. If a system experiences an increased data input, additional hardware resources (data nodes) can be easily added to a cluster to handle the increased workload. For more details on the architecture of Hadoop and MapReduce, readers can consult [119, 120].

Constraints. This tactic assumes that additional hardware is available for horizontal scaling of a system. Furthermore, the introduction of overhead introduced due to disk read/write operations is a pressing issue with MapReduce, which can significantly prolong the response time of a BDCA system.

Example. MapReduce tactic is used by a number of the reviewed systems, however, we describe a few of them to illustrate its contribution in achieving scalability.

- Traffic Measurement and Analysis with Hadoop [S70]: This system uses heuristic algorithm [121] that enables mappers to read packet records from HDFS based on timestamp bit of a packet header. In order to evaluate scalability, 1 TB of security event data is analysed by a cluster with varying number of nodes from 5 to 30. It is observed that system performance improves in proportion to the resource allocation. For example, the analysis completion time decreases from 71 minutes to almost 36 minutes as the number of nodes increases from five nodes to 10 nodes.
- IDS-MRCPSO [S31]: This system uses Particle Swarm Optimization clustering algorithm [122] with MapReduce. To investigate scalability, the system is implemented with different number of nodes. It is found that the speedup time increases linearly in the start from 2 to 8 nodes, but it starts to diverge from linear while moving from 8 to 16 nodes. This diversion is attributed to Hadoop framework i.e., starting MapReduce jobs and storing intermediate results.
- Extreme Learning Machine [S24]: This BDCA system combines the linear scaling capability of Extreme Learning Machine algorithm [95] and MapReduce. The system has been implemented with 15, 20, 25, and 30 nodes to evaluate its scalability. The system shows linear scaling from 15 to 25 nodes but diverges from linear scaling after 25 nodes. The authors argue that such diversion is experienced due to the increased communication between the nodes.

Dependencies. As such this tactic does not require any other tactic for its implementation.

2.5.4 Reliability

This section reports the architectural tactics that are related to Reliability quality attribute.

2.5.4.1 Data ingestion monitoring

Introduction. Data Ingestion Monitoring tactic is found in Streaming-based Threat Detection System [S1], Malicious IP Detection [S18], and GPGPU-based IDS [S35]. This tactic monitors the flow of data from data collector into the computing servers in real-time data streaming systems. If the speed of data influx becomes too fast to be handled by a computing server, this tactic automatically blocks the flow of data into that server.

Motivation. Security systems are expected to receive and handle a large amount of security event data. However, sometimes the security event data might be generated and collected at a speed that is beyond the capacity of a computing cluster. Such a situation can lead the server to crash. Therefore, a tactic is required to monitor the speed of data generation and control the flow of data into the computing servers.

Description. The main components of Data Ingestion Monitoring tactic are shown in Figure 2.22. The *data collector* component collects data from various sources using different tools such as Wireshark for collecting network traffic data. The data collected by various nodes of the *data collection* component is moved to the *distributed data storage and analysis* component through *data ingestion monitor*. The role of *data ingestion monitor* is to keep the real-time data streaming into the *distributed data storage and analysis* cluster. The *data ingestion monitor* monitors the speed of the flow of the data and when it becomes higher than the capacity of the computing server, this tactic will block the flow of the data into the computing cluster. The *data ingestion monitor* adjusts the flow of the data between *data collection* component and *distributed data storage and analysis* component. The stream of security event data fed into the *distributed storage and analysis* component is analysed for detecting cyber attacks.

Constraints. Data Ingestion Monitoring Tactic requires expertise for installation of data ingestion monitor, which is a challenging task, especially in highly distributed setup. Furthermore, the tactic also requires more investment for setting up a monitoring server. The tactic is best fit for BDCA systems that are deployed in enterprises, which generate a large volume of security event data at a high speed and stream it directly into the analytic component for real-time processing.

Example. Data Ingestion Monitoring tactic is incorporated in the following systems.

- Malicious IP Detection [S18]: This system demonstrates how Data Ingestion Monitoring tactic prevents computing server from crashing. Flume server[10] is used as data ingestion monitor to monitor and control the influx of data into the computing cluster. The authors have evaluated the system with four data influx rates (i.e., 0.4 million records/min, 0.7 million records/min, 0.8 million records/min, 0.85 million records/min). It has been reported that the computing server has idle capacity up to 0.8 million records/min, however, as the speed is increased to 0.85 million records/min, the computing server reaches its maximum limit. Increasing the data influx rate beyond 0.85 million records/min would crash a computing server. Therefore, at 0.85 million records/min, the data ingestion monitor controls the speed from increasing.

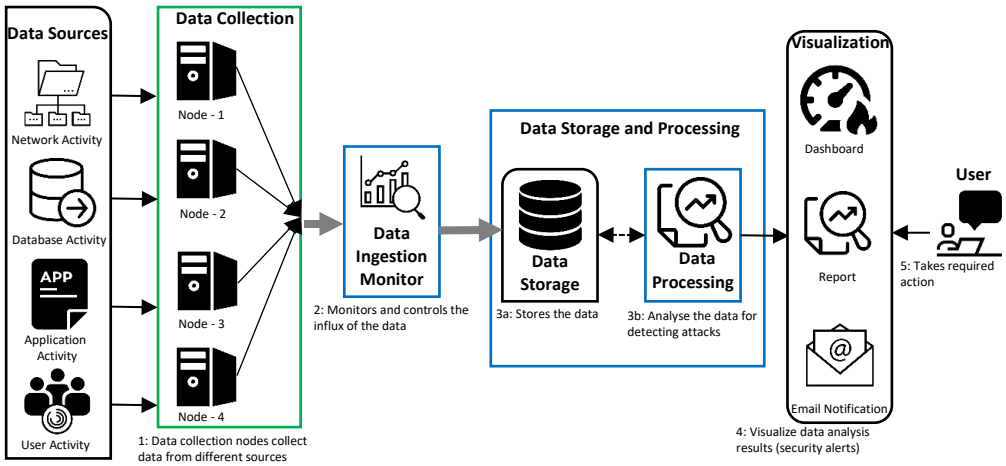


Figure 2. 22 Data ingestion monitoring tactic

[10] http://flume.apache.org/

- Streaming-based Threat Detection System [S1] and GPGPU-based IDS [S35]: Both these systems implement Data Ingestion Monitoring tactic through Flume service; however, these studies have not evaluated the impact of the tactic on the reliability of the system.

Dependencies. Data Ingestion Monitoring tactic does not depend upon any other tactic; however, it can better be consolidated when implemented together with Secure Data Transmission tactic (Section 2.5.5.1), which will help get the data to be monitored in its original form.

2.5.4.2 Maintaining multiple copies

Introduction. Maintaining Multiple copies tactic can be found in any system that leverages Hadoop framework, however, only seven [S2], [S6], [S10], [S13], [S17], [S31], [S71] of the reviewed papers explicitly demonstrate how this tactic can make a system more reliable. This tactic enables Hadoop's HDFS to maintain multiple copies of each data block. These copies are placed on different nodes in a Hadoop infrastructure. In case of a node failure, the computation is diverted to another node that hosts the copy of the data block [123].

Motivation. During data processing, a Hadoop node may go down or fail due to several reasons such as RAM crash, power shutdown, and hard-disk failure. In such a case, data on the same node will no longer be accessible and a user has to wait until the issue is addressed. Therefore, a mechanism is required that can enable a BDCA system to have access to the data even if a node fails.

Description. The main components of Maintaining Multiple Copies tactic are shown in Figure 2.23 with the numbers showing the sequence of operations. The *data collector* component collects security event data from different sources, which is stored in files. The files are divided into blocks of same sizes. The default block size is 64 MB; however, the block size can be customized. For the sake of understanding, only one file split into four blocks is shown in Figure 2.23. After dividing files into blocks, the blocks are stored on different nodes (i.e., machines). By default, three copies of each data block are created. However, the number of replicas created can be configured. Each data block is stored on three different machines. For example, Block 1 is stored on Node 1, Node 4, and Node 6. Then the *data processing* component reads the data blocks for security analysis. If a data block is not accessible at any instance (for instance block 1 on Node 1), the same data block will be accessed from another node (for instance

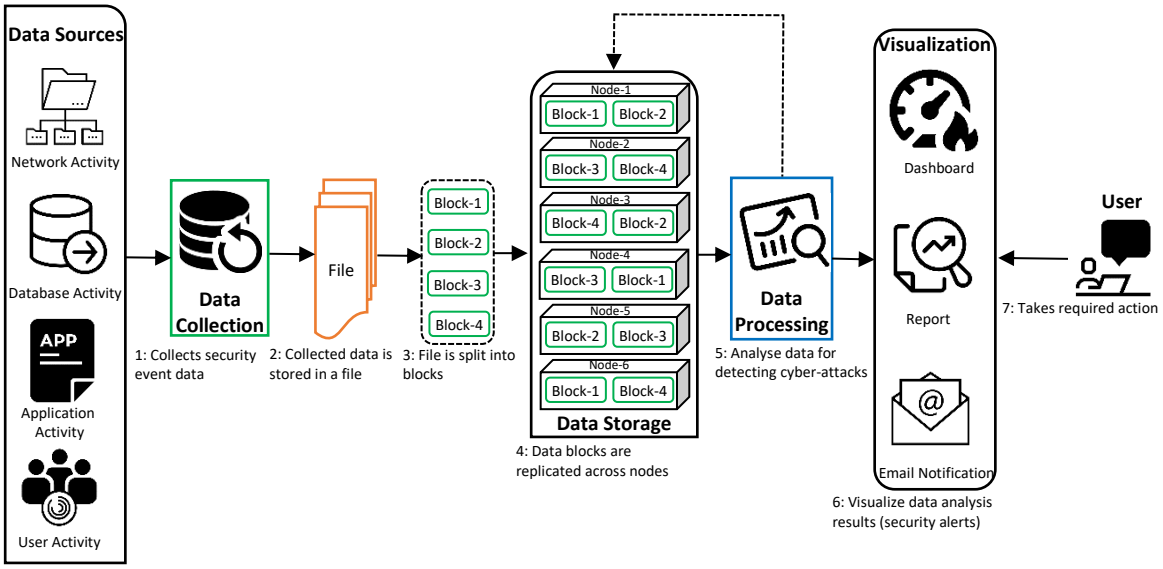


Figure 2. 23 Maintaining multiple copies tactic

block 1 from Node 4). Once the data is analysed in a reliable manner, the results are shared with the user through the *visualization* component.

Constraints. With the incorporation of Maintaining Multiple Copies tactic, the cost of storing the same data increases. This tactic may also introduce some data inconsistencies.

Example. Following are a couple of examples that illustrate the implementation of this tactic.

- Honeypot-based Phishing Detection [S17]: The impact of replication factor (i.e., number of copies) on the performance of a BDCA system has been investigated with one, two, and three replicas. It is observed that a system's response time is best with three replicas (21 min), followed by two replicas (22 min). With one replica, a system can have the lowest response time (25 min) and no reliability in case of a node failure. The difference in response time is due to the number of non-local accesses incurred in each case, which is 4 with three replicas, 7 with two replicas, and 113 with one replica.
- Reliable Traffic Analysis [S71]: This system has been tested with two node failure scenarios – one where node executing map task fails and another where node executing reduce task fails. The node running map task is forced to reboot, while the node running reduce task is shutdown. The corresponding map tasks and reduce tasks are migrated to other nodes having the replicas of the data. It has been observed that the system remained available, however, it takes little more time in completing the tasks. For example, for flow count of 3.2 million, the completion time without nodes failure is 220.2 seconds, while with node failure the completion time is 380.2 seconds.

Dependencies. Maintaining Multiple Copies tactic does not require any other tactic for its implementation; however, it can better be consolidated when implemented together with Secure Data Transmission tactic (Section 2.5.5.1), which will help get the data to be replicated in its original form.

2.5.4.3 Dropped NetFlow detection

Introduction. Dropped NetFlow Detection tactic has been found in Hybrid Stream and Batch Analyzer [S54]. NetFlow is a sequence of packets that typically share seven values, which include destination IP, source IP, destination port, source port, IP protocol, type of service, and ingress interface [85]. This tactic helps BDCA systems, which rely on collecting and analysing NetFlow data for detecting attacks, to detect if the data collector has missed some NetFlow. The tactic monitors the NetFlow Sequence Numbers [124] and if they are found out of order, a warning message specific to that stream is logged. This tactic is important for detecting faults that can have severe consequences for reliable data collection.

Motivation. For BDCA system relying on NetFlow data, it is important to collect and analyse each NetFlow. Missing a single NetFlow may result in missing the detection of an attack (e.g., a single flow record may summarize malicious transfer of a large amount of data). In the existing network dynamics, it is quite possible that some NetFlow are missed due to several reasons that include (a) network dropping packets; (b) router not able to handle the volume of traffic; and (c) data collector of a BDCA system not able to handle the volume of traffic. Hence, it is quite crucial to monitor and detect whether a BDCA system is analysing all NetFlow or there are some NetFlow going missing. In the latter case, a security administration needs to trace the source causing loss and fix the issue.

Description. The main components of the Dropped NetFlow Detection tactic are shown in Figure 2.24 along with the number indicating the sequence of the operations. The network traffic is passing

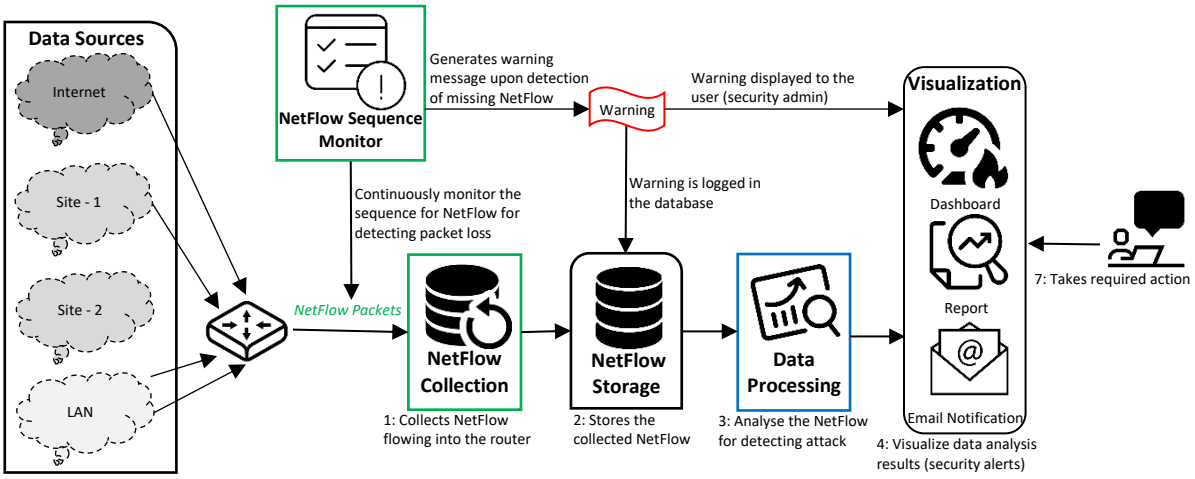


Figure 2. 24 Dropped NetFlow detection tactic

through the router shown in the figure. A *NetFlow collector* is connected to the router, which collects the NetFlow and stores them in the *NetFlow storage*. During the NetFlow collection process, *NetFlow sequence monitor* component is monitoring the sequence numbers embedded in the NetFlow. If the sequence numbers are found out of order at any stage, the *NetFlow sequence monitor* generates a *warning message* indicating the missing flow in the particular stream of NetFlow. The warning message is logged alongside the specific stream in the *NetFlow storage* to specify that the stream of NetFlow has some flows missing that might be crucial in relation to detecting an attack. At the same time, a warning is displayed to a security administrator through the *visualization* component. Then a security administrator may take the required actions for fixing the issue due to which some NetFlow may get dropped.

Constraints. The NetFlow sequence monitoring should cope with the speed at which NetFlow is collected so that additional delay for monitoring can be avoided. This tactic is best fit for BDCA systems that monitor highly busy networks where there is a chance of packet loss. It is worth noting that this tactic only detects the loss of NetFlow. For mitigation, i.e., identifying the source of loss and fixing, it requires a separate mechanism.

Example. NetFlow is a network protocol developed by Cisco for collecting network information (e.g., source IP, destination IP, and size of network traffic) for the purpose of monitoring network activities. Hybrid Stream and Batch Analyzer [S54] is the example system that have implemented Dropped NetFlow Detection tactic. The system has multiple NetFlow collectors that run on the probe nodes. Each NetFlow collector has two associated AMPQ queues [125] – one for the data stream and another for log information. The system experiences loss of NetFlow due to link aggregation. In order to detect the missing NetFlow, the sequence numbers of the NetFlow are monitored and when they are found out of sequence, a warning message specific to the stream is placed in the log information queue.

Dependencies. This tactic can be implemented independent of any other tactics.

2.5.5 Security

This section reports the architectural tactic that is related to Security quality attribute.

2.5.5.1 Secure data transmission

Introduction. Secure Data Transmission tactic has been reported in Cloud-based Threat Detector [S10] and PKI-based DIDS [S73]. This tactic ensures secure transmission of security event data from data collection nodes to the data processing nodes.

Motivation. A BDCA system protects a critical infrastructure from cyber attacks, however, it also requires some security measures for itself. In most cases, the data collection and analysis operations reside on separate nodes (physical machines). The nodes hosting data collection operation are placed in different regions within an organization [126]. The data collected by these nodes is transmitted to the nodes (cloud cluster) hosting the data analysis operation. It is quite possible that an attacker intercepts this data transmission (e.g., with a sniffing tool) to tamper with or spoof the data [127]. The consequences of such interception can be catastrophic if an attacker manages to tamper with the data critical for detecting an attack. Therefore, it is important to guarantee secure transmission of data from data collection nodes to the cloud cluster responsible for data analysis.

Description. Figure 2.25 shows the main elements of the Secure Data Transmission tactic. The nodes for collecting security event data are placed in different regions for collecting different types of data. Some collect network traffic and others collect database access information. Security measures are applied to the collected data to ensure its security while in transition from one component to another component. Some systems prefer to encrypt the collected data and then transfer the data. Other systems use Public Key Infrastructure (PKI) to ensure secure exchange of data and verification of the party sending the data. As data is received by the *data storage and analysis* component in a secure way, the data analytics operations are applied to analyse the data for detecting attacks. The results of the analysis are presented to users through the *visualization* component.

Constraints. The introduction of security measures in communication among different components will slow down the data transmission process and eventually the response to an attack. It is also worth noting that Secure Data Transmission tactic protects data only in motion state, which means that additional measures will be required to secure the data in use and in rest state both at the collection nodes and the storage and analysis nodes. For instance, the injection of false data in one of the data collection nodes can have disastrous consequences in terms of attack detection.

Example. The two systems that have implemented Secure Data Transmission tactic incorporate some kind of security measure for securing the data during transition.

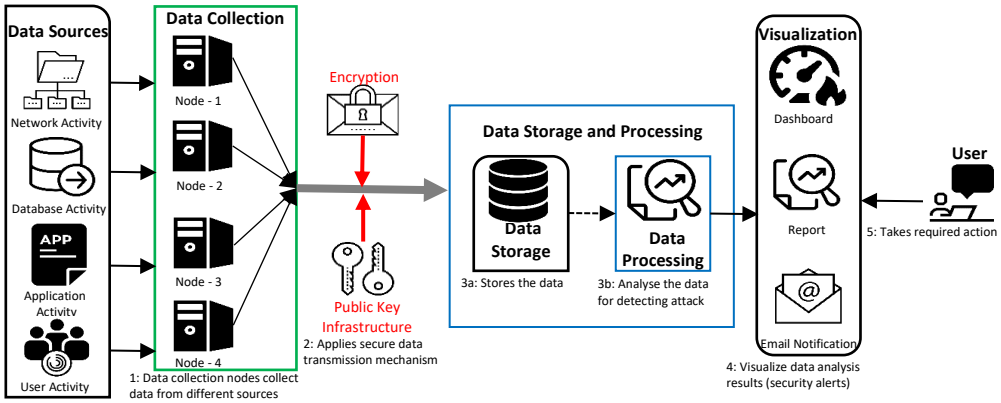


Figure 2. 25 Secure data transmission tactic

- Cloud-based Threat Detector [S10]: In this system, all communications between various modules take place in an encrypted form, which is achieved by adopting HTTPS protocol. In order to ensure the authenticity of access, session tokens are checked and verified.
- PKI-based DIDS [S73]: This system deploys Public Key Infrastructure between data collection nodes and data storage and analysis nodes. According to this security measure, digital certificates are used to encrypt data to be transmitted from data collection to data storage and analysis. It also enables the data collection and data storage component to verify the authenticity of data collection nodes. This way, even if an attacker intercepts the transmission, it will be very hard to tamper with the confidentiality or integrity of the data.

Dependencies. This tactic can be implemented independent of other tactics.

2.5.6 Usability

This section reports the architectural tactic that is related to Usability quality attribute.

2.5.6.1 Alert ranking

Introduction. Alert Ranking tactic has been reported in Hunting attacks in dark [S41]. This tactic ranks the alerts generated by a BDCA system based on the dangerousness of alert. This ranking improves the usability of a system by enabling users (e.g., security administrator) to respond to alerts on a priority basis.

Motivation. A BDCA system can generate a large number of alerts [128, 129]. In around 90% of the cases, the alerts are either false positives or of low importance [130]. It is also well understood that the more dangerous an alert is the more challenging it is to respond and mitigate its effects [S41]. If a security administrator responds to alerts in a sequence in which alerts are generated, it is quite likely that serious alerts requiring an immediate response will be responded quite late. On the other hand, if a security administrator assesses the seriousness of alerts, it will be quite time-consuming. A delayed response may be quite detrimental as it will be challenging to mitigate the effects of an attack. Therefore, it is important to incorporate a mechanism in a BDCA system that can help a security administrator to prioritize the responses to more serious alerts.

Description. The main components of Alert Ranking tactic are shown in Figure 2.26 with the numbers indicating the sequence of the operations. The *data collection* component collects security event data from different sources, which is pre-processed by the *data pre-processing* component. The pre-processed security event data is forwarded to the *data processing* component, which analyses the data for detecting

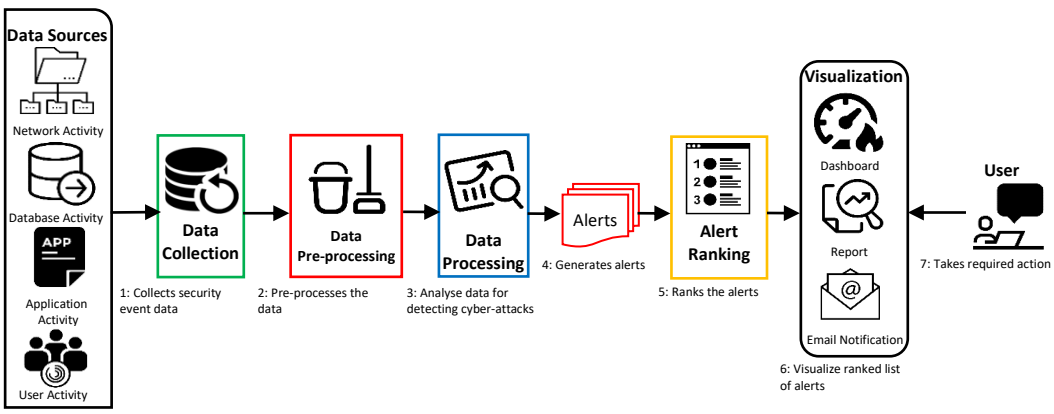


Figure 2. 26 Alert ranking tactic

cyber attacks. The results of the analysis (i.e., alerts) are forwarded to the *alert ranking* component, which ranks the alerts based on a predefined criterion to assess the impact of an alert on the overall organization's infrastructure. The criterion for ranking alerts depends upon an organization. For example, the ranking criteria for an organization vulnerable to DoS attacks will be different to an organization vulnerable to brute force attacks. The ranked list of simple and easy-to-interpret alerts is shared with security administrators through the *visualization* component, which eases the job of a security administrator to first respond to the alerts on top of the rank list.

Constraints. Alert Ranking tactic requires that the functionality responsible for ranking the alerts should not be computationally expensive, otherwise it will cancel the benefit of quick response acquired through prioritized response to dangerous alerts. Moreover, all ranking algorithms are not equally accurate [131]. Therefore, it is important to carefully select the ranking algorithm (criteria) that can accurately rank alerts based on the specific security requirements of an organization.

Example. Hunting attacks in the dark [S41] is an example system that has implemented Alert Ranking tactic. This system ranks an alert based on the amount of traffic that belongs to alert. The underlying consideration for this criterion is that alerts indicating a sudden increase in the network traffic (e.g., flooding-based attacks) are the most serious. The system uses the number of packets and the number of bytes belonging to each alert to calculate Dangerousness Index (DI) for each alert using the formula (i.e., DI(alert) = Number of packets + Number of bytes). The bigger the DI of an alert is, the more dangerous is the alert. Based on the DI scores, alerts are ranked and presented to security administrators.

Dependencies. Alert Ranking tactic does not entirely depend on any other tactic; however, this tactic is usually implemented together with tactics that can support the data processing component such as Parallel Processing tactic (Section 2.5.1.4) and Attack Detection Algorithm Selection tactic (Section 2.5.2.3).

2.6 Discussion

Section 2.3, Section 2.4, and Section 2.5 have presented the findings of this chapter. In this section, we draw some lessons from the findings and reflect upon the potential usefulness and implications of the findings for practitioners.

2.6.1 Lessons Learned

2.6.1.1 Mapping of tactics to the modules of a BDCA system

Figure 2.27 shows the mapping of the tactics reported in Section 2.5 onto the major modules of a BDCA system: *data collection, data storage, data pre-processing, data processing, data post-processing and visualization*. The mapping is expected to help a reader to understand which modules(s) or stage(s) are highly critical from an architectural perspective. It is clear that around 47% (i.e., 8 out of 17 tactics) of the architectural effort is focussed on the *data processing* module. The *data processing* module mainly relates to accuracy (four tactics) and response time (three tactics), which are considered as the two highly emphasized quality attributes (*Lesson 3*). We recommend that a significant amount of time and energy be invested in designing this module. A flawed design decision can lead to detrimental consequences. For example, the selection of an inefficient ML algorithm (Section 2.5.2.1) can negatively impact a BDCA system's several qualities attributes such as accuracy and response time ([S1], [S9], [S10], [S14], [S27]). The mapping also shows that the *visualization* module's design has received the least

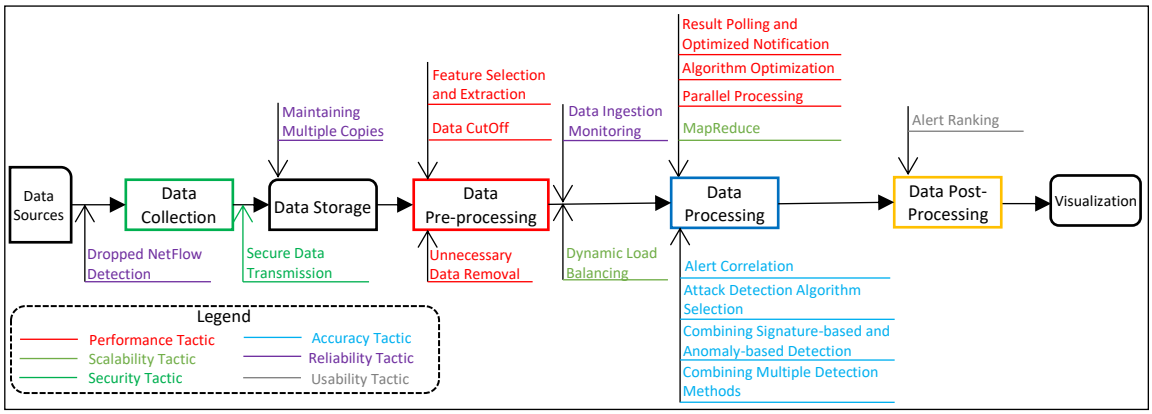


Figure 2. 27 Mapping of tactics to the modules of BDCA system

amount of attention in the reviewed systems. Given the importance of efficient visualization in big data systems [132], there is an urgent need of advanced research aimed at realizing the potential of *visualization* for BDCA systems.

> ***Lesson 1:*** *From an architectural perspective, Data Analysis is the most critical module of a BDCA system*

2.6.1.2 Mapping of quality attributes to the modules of a BDCA system

The mapping (shown in Figure 2.27) provides a relationship between the quality attributes and different modules. This type of visual information is expected to help a reader to understand which modules are important for achieving certain quality attributes. Figure 2.27 shows that the accuracy related tactics (i.e., Section 2.5.2) are completely focussed on the *data processing* module. The tactics associated with the remaining quality attributes are largely scattered throughout a system. For example, response time related tactics (Section 2.5.1) are incorporated in the *data pre-processing* and *data processing* modules. The reliability related tactics (Section 2.5.4) are associated with multiple modules such as *data collection*, *data storage*, and *data processing*. This analysis identifies the areas of knowledge/expertise that an organization aspires to have for its desired quality goals. For instance, a BDCA system having accuracy as the primary quality goal requires expertise in data analytics. An organization intending to develop a BDCA system that can satisfy multiple quality goals should have a diverse set of resources and expertise ranging from data collection to data analytics and visualization.

> ***Lesson 2:*** *Apart from accuracy, the architectural support for achieving quality goals is largely distributed among the modules of a BDCA system*

2.6.1.3 First class quality attributes

Given that the priority of quality attributes of a system varies from domain to domain [33], it is important to know the priorities of quality attributes for BDCA systems from our findings (*RQ1*). Our findings indicate that a wide range of quality concerns have been reported in the reviewed papers – ranging from *performance* to *stealthiness* (Table 2.6). We observe a significantly highly number of papers focus on three quality attributes: *response time* (90.5%), *accuracy* (58.1%), and *scalability* (54%). Our finding related to architectural tactics (*RQ2*) further strengthen this observation, where 12 out of 17 tactics (i.e., 70.5%) aim to achieve these three quality attributes. Ian and colleagues' paper [66] also emphasize *response time* and *scalability* as key quality attributes for a big data analytics system. A

possible reason for not including accuracy as a key quality attribute could be the application domains (i.e., aeronautics and healthcare) considered in [66], where the cost of misclassification may not be as detrimental as in cyber security [63]. Given that it is challenging to achieve several quality attributes for a system, a good approach is to first focus on the most significant ones (e.g., *response time*, *accuracy*, and *scalability*).

> ***Lesson 3:*** *Response time, accuracy, and scalability are the three most important quality attributes in the context of BDCA.*

2.6.1.4 Under-addressed quality attributes

Section 2.4 reports our analysis of the importance of specific quality attributes for BDCA systems. Apart from response time, accuracy, scalability, reliability, security, and usability, our analysis has revealed that quality attributes such as interoperability, adaptability, modifiability, generality, stealthiness and privacy assurance are also important (Table 2.6) for BDCA systems. There is a little focus on support for achieving these quality attributes in the reviewed papers. It is worth mentioning that some of the reviewed papers briefly mention the need of achieving these qualities. For example, the papers [S19] and [S42] advocate the use of open source tools for achieving interoperability. The papers [S10] and [S29] recommend higher modularity for achieving interoperability and modifiability. For generality, the papers [S12] and [S23] encourage the incorporation of diverse data sources and frequently updating attack patterns. However, the reported recommendations are quite generic and abstract. Therefore, we assert that there is a need for more research efforts aimed at providing support for interoperability, modifiability, adaptability, generality, stealthiness and privacy assurance in BDCA systems. Such a support is expected to devise tactics, patterns, and design principles for addressing quality concerns related to the aforementioned quality attributes.

> ***Lesson 4:*** *It is crucial to investigate support for achieving interoperability, modifiability, adaptability, generality, stealthiness, and privacy assurance in BDCA systems*

2.6.1.5 Tactics' evaluation

It is important to assess the system-wide impact of the elicited architectural tactics for BDCA systems because (i) the tactics have been elicited from specific implementations, (e.g., BDCA system detecting DoS attack) using different datasets (e.g., KDD, DARPA, and CAIDA), therefore, their incorporation and potential impact cannot be generalized to all types of BDCA systems (ii) Due to the lack of sufficient empirical evidence, a software architect may employ a tangled implementation of a tactic(s), which can lead to severe negative impacts [133] (iii) a concrete catalogue of qualitative and quantitative evidence will build software architect's trust in the tactics and will highlight the cautions considerable during the incorporation of various tactics. The primary studies from which the tactics have been extracted do include an evaluation phase, however, it does not report the evaluation of each of the identified tactics. For example, the impact of Combining Multiple Detection methods tactic (Section 2.5.2.4), Alert Correlation tactic (Section 2.5.2.1), Dynamic Load Balancing tactic (Section 2.5.3.1), Dropped NetFlow Detection tactic (Section 2.5.4.3), Secure Data Transmission tactic (Section 2.5.5.1), and Alert Ranking tactic (Section 2.5.6.1) have not been evaluated. We believe that there can be expert based assessment (e.g., [134]) or experimentation (e.g., [135], [136]) to qualitatively and quantitatively evaluate the potential impact of the identified tactics.

Lesson 5: *It is essential to (qualitatively and quantitively) investigate the system-wide impact of the codified tactics*

2.6.1.6 Quality trade-offs among tactics

It is well known that architectural tactics may positively support one set of quality attributes but may have negative impact on another set of quality attributes [33]. For example, Alert Correlation tactic (Section 2.5.2.1) helps achieve *accuracy* but can have negative impact on *response time;* Maintaining Multiple Copies tactic (Section 2.5.4.2) improves *reliability* but increases the overall *storage cost*. Moreover, an architectural tactic can have a positive impact on multiple quality attributes unlike its association with one quality attribute. For instance, Parallel Processing tactic (Section 2.5.1.4) improves both *response time* and *reliability*. Making and understanding the trade-offs centric design decisions can be quite challenging task [33]. We present a two-dimensional matrix (Figure 2.28) to highlight the relationship between the reported advantages/disadvantages of the quality attributes and the codified tactics. The matrix can be used by a software architect in two ways. First, a software architect can identify which benefits are of key interest in their situation and accordingly select the set of tactics. For example, a BDCA system aims to detect highly sophisticated attacks should incorporate Alert Correlation (Section 2.5.2.1) and Combining Multiple Detection Methods (Section 2.5.2.4) tactics. Second, the matrix helps in assessing the impact of tactics at the design time. For instance, the matrix illustrates that Secure Data Transmission tactic (Section 2.5.5.1) improves data integrity and data confidentiality, however, slows down the overall response time. This information can stimulate a software architect to deliberate what is more significant (i.e., data security or quick response) for a particular scenario. The study report in this chapter has highlighted the key quality attributes, the architectural tactics, and the quality trade-offs related to the architectural tactics. An interesting avenue for future investigation is the applicability of multi-objective optimization [137] for evaluating the effectiveness of architectural tactics with respect to conflicting quality requirements.

Lesson 6: *It is important to analyze the quality trade-offs among the codified tactics at the design time before selecting/dropping a tactic*

2.6.1.7 Dependencies among tactics

The reported tactics can be used in different stages of a system (Figure 2.27). It is important to understand how different tactics work together by exploring the inter-dependencies and collaborations among the reported tactics. As mentioned in Section 2.5, architectural tactics may depend on other tactics for achieving their respective objectives. For example, Feature Selection tactic (Section 2.5.1.3) depends upon Parallel Processing tactic (Section 2.5.1.4) for speeding up the process of feature selection. A combination of tactics may complement each other in a way that allows an architect to drop one tactic in favour of another tactic to resolve a conflict between two quality attributes. For example, Combining Multiple Detection Methods tactic (Section 2.5.2.4) and Signature-based Detection tactic (Section 2.5.2.2) complement each other to achieve *accuracy* quality attribute. However, the use of both tactics will increase *response time*. Whilst the dependencies among tactics have been reported to a certain level in Section 2.5, this review has enabled us to assert there is an important need of empirical research to identify and understand the deeper dependencies among the identified architectural tactics.

Lesson 7: *Research is essential to investigate the (level of) dependencies among the codified tactics*

Quality Advantages / Architectural Tactics / Quality Disadvantages	ML Algorithm Optimization	Unnecessary Data Removal	Feature Selection and Extraction	Parallel Processing	Result Polling and Optimized Notification	Data CutOff	Alert Correlation	Combining Signature-based and Anomaly-based Detection	Attack Detection Algorithm Selection	Combining Multiple Detection Methods	Dynamic Load Balancing	MapReduce	Data Ingestion Monitoring	Maintaining Multiple Copies	Dropped NetFlow Detection	Secure Data Transmission	Alert Ranking
Reduce computational complexity	✓		✓						✓								
Removes unnecessary data		✓															
Remove redundant records		✓															
Dimensionality reduction			✓														
Reduce data complexity			✓														
Distributes processing among nodes				✓								✓					
Reduce processing delays					✓												
Applies customizable data cutoff						✓											
Reduce false positives							✓	✓		✓							
Enable detection of highly sophisticated attacks							✓			✓							
Enable detection of zero-day attacks							✓	✓									
Ensures multi-dimensional data analysis										✓							
Balances load among the nodes											✓	✓					
Improve communication among nodes												✓					
Improves synchronization among tasks												✓					
Controls influx of data													✓				
Handles node failures													✓				
Improves data integrity														✓	✓	✓	
Detect data collection faults															✓		
Improves data confidentiality																✓	
Facilitates quick response																	✓
Risk of discarding relevant data		✓				✓											
Risk of discarding critical features			✓														
Potential of damage to logical records				✓													
Increase in computational cost				✓													
Risk of evading attack detection						✓											
Increase in computational time							✓	✓		✓		✓					
Requires domain knowledge							✓	✓									
Introduce interoperability challenges																	
Increase storage cost														✓			
Introduce data inconsistencies														✓			
Slows down overall response time																✓	✓
Requires more resource and expertise													✓				

Figure 2. 28 Matrix linking quality advantages/disadvantages to architectural tactics

2.6.1.8 Modelling the tactics

The architectural tactics, as reported in this study, are not modelled using a standard modelling language such as Unified Modelling Language (UML). However, we assert that it would be useful to define the semantics of the codified tactics (similar to [138]) using a standard modelling language. Our assertion is motivated by two reasons – (i) Modelling of a tactic is expected to abstract the explicit design details to enable a software designer to focus on the significant tasks of a tactic in a designed solution (ii) Modelling is expected to ensure that a tactic's constraints are respected and to efficiently trace the tactics in the source code. A tactic's modelling is expected to capture the structural and behavioural details of the tactics. For example, the model for Features Selection tactic (Section 2.5.1.3) should illustrate the roles of feature selection and feature extraction components, the details of their interaction with other components, and the lifeline of their interactions. It is important to model the dependencies and relationships among tactics. Such modelling is meant to reveal the type of dependencies among tactics. For example, a software architect needs explicit guidance on whether the Parallel Processing tactic (Section 2.5.1.4) is mandatory for Feature Selection tactic (Section 2.5.1.3) or it

is only suggestive. One such option for modelling the relations among tactics is Features Modelling [139], which is a technique for modelling the commonalities and dependencies among patterns and tactics.

> ***Lesson 8:*** *Formal modelling of the codified tactics is required to facilitate the software architect in better understanding the role of tactics and respect the associated constraints*

2.6.1.9 Meta-data analysis lessons

Our demographic view (Section 2.3) has revealed some points that should be of interest and value to researchers and practitioners working in the area of BDCA. The popularity and adoption of Spark (as compared to Hadoop) is quite evident from Figure 2.6. One possible reason for this, as reported in several of our reviewed papers (e.g., [S16], [S17], and [S29]), is the processing speed with respect to which Spark outclasses Hadoop by at least 10 times. Section 2.3.2 reveals that only four papers out of 74 included the papers that were published in software engineering related venues; this situation indicates a lack of focus on this important domain of research and development. This paucity is not limited to just BDCA, instead circles the focus of software engineering for any domain of big data analytics as highlighted by Madhavji and colleagues [140]. This study has revealed several research areas (reported in Section 2.6.1.4, Section 2.6.1.5, Section 2.6.1.6, Section 2.6.1.8) that requires the attention of SE research community. Our finding (Figure 2.4) also strengthens the already proposed view [15] that the traditional cyber security solutions will not be able to sustain for long, hence, the adoption of big data technologies is inevitable as evidenced by its increased adoption over the years.

> ***Lesson 9:*** *In the context of BDCA (i) Spark is getting more popular as compared to Hadoop (iii) there is a paucity of Software Engineering focus and (iii) Role of big data technologies is on rise.*

2.6.2 Implications

Given the focus of our study (i.e., architectural tactics) and its intended application for practitioners, we make a few recommendations for practitioners about using the codified tactics.

2.6.2.1 Choosing among competing tactics

Our catalogue of tactics, a kind of design space, can be used by practitioners for supporting different types of quality attributes in BDCA systems. For example, there are six tactics available (as shown in Figure 2.8) that aim to achieve *response time* in different ways. Considering the quality trade-offs (Lesson 6), it is not a good design decision to randomly select any particular set of tactics [33]. The choice of which tactic(s) to choose or not to choose becomes a challenge for software designers. Section 2.5 reveals extensive correlation and dependencies among the codified tactics (i.e., including or excluding a tactic(s) impacts other tactics and subsequently quality attributes), which should be selected and used through a systematic analysis and design process [33]. Whilst there is no silver bullet for optimal choice of tactics, we suggest prioritizing the selection of tactics, which has minimum impact (as highlighted in the *consequence* section for each tactic) on other quality attributes. The topic of tactic selection for designing BDCA systems is further discussed in Chapter 4. Future research should empirically investigate the system-wide impact of the identified tactics (i.e., Lesson 5). Such research can help practitioners to devise suitable strategies for choosing an optimal set of tactics for design decisions.

Recommendation 1: *Follow a comprehensive analysis and design process to choose among competing tactics*

2.6.2.2 Refining and contextualizing tactics

The identified tactics have been codified at different abstraction levels. For example, Feature Selection tactic (Section 2.5.1.3) is at a higher level of abstraction, which offers flexibility in its implementation. For instance, a software architect can decide whether features should be locked at the design time or runtime and which technique should be used for feature selection. On the other hand, tactics such as *Result Polling and Optimized Notification tactic* (Section 2.5.1.6) lies at a lower abstraction level, where an architect has little (if none) flexibility during implementation. That means the abstraction level for several tactics (e.g., Alert Correlation, Combing Multiple Detection Methods, and Data Ingestion Monitoring) is quite high; hence, these tactics need to be refined according to specific functional and non-functional requirements. The use of the identified tactics should take context (e.g., resource and cast) into consideration. For example, the availability of enough computational resource can support Feature Selection tactic (Section 2.5.1.3) in a way that features can be selected intelligently at runtime, which will add to accuracy without impacting the response time. Similarly, a software architect has several options [141] for employing Alert Correlation tactic (Section 2.5.2.2). The refinement and contextualization of tactics is further exemplified in Chapter 3, where tactics are implemented as per the underlying functional and non-functional requirements. For example, InfoGain technique [142] is used for the implementation of Feature Selection tactic as this technique is quite useful (in terms of accuracy and efficiency) for the selection of features from a lower dimensional data. Given that the security datasets we used are of a lower dimension, we opted to use InfoGain for implementing the Feature Selection tactic.

Recommendation 2: *Refine and contextualize tactics according to the underlying requirements and resource*

2.6.2.3 Tools Literacy

The reviewed papers reported a variety of big data tools used in different phases of BDCA systems – *data collection* (e.g., Wireshark and nfdump), *data pre-processing* (e.g., Fume and Kafka), and *data analysis* (e.g., ML algorithms and big data frameworks). It is also evident that these tools vary from each other in terms of their performance and operational requirements. For example, [S17] reports that Apache Spark is ten times faster than Apache Hadoop, however, Apache Spark requires high memory as data is stored in memory instead of disk. Therefore, the option of having such a large pool of tools makes it challenging as to which set of tools should be selected. Apart from a few papers (e.g., [S9], [S14], [S26]) that compare the performance of some ML algorithms, there is a scarcity of literature aimed at comparing the relative performance of various tools (e.g., big data frameworks). We assert the need for comparative studies that can give a consolidated view for the selection of big data tools for BDCA. We recommend that software architects of BDCA systems should take a broader view of big data tools from multiple angles (e.g., cost, team and expertise) before incorporating a particular tool in a BDCA system. In this regard, referring to some existing literature (e.g., [143, 144]) for the selection of tools can be helpful.

Recommendation 3: *Take time to make well-informed decisions about the selection of tools for BDCA*

2.7 Related work

The design and development of BDCA systems have been drawing an increasing amount of attention of researchers and practitioners. There have been a few literature reviews on security analytics from different perspectives. Whilst there has been no literature review of architectural aspects (i.e., quality attributes and architectural tactics) of BDCA systems; nor has there been any review conducted using Systematic Literature Review methodology. We briefly summarize some of the existing reviews on BDCA to position our work. Kaj et al. [145] report a literature survey that provides a taxonomic analysis of the techniques for cyber security analytics. They categorized the techniques into descriptive, diagnostic, visual, predictive, and prescriptive. The prescriptive techniques have the potential for further research. Lidong and Randy [146] reviewed the literature on the big data processing frameworks (e.g., Spark and Storm), data pre-processing, and machine learning for security analytics. However, their work does not address the architectural aspects.

Tariq and Uzma [143] reviewed the literature that sheds light on different types of cyber attacks (e.g., botnets and phishing), various sources of security event data, sample outputs of a security analytics system, and commercially available security analytics solutions. Jeong et al. [147] investigated various challenges and Hadoop based potential solutions for IDS. The reported challenges include emerging attack patterns, high implementation cost, and large volume, high velocity and heterogeneous nature of security event data. The authors argue that the use of Hadoop, MapReduce and NoSQL databases, and incorporation of open source software help address the aforementioned challenges. Rasim and Yadigar [148] reviewed the state-of-the-art big data technologies for security analytics. Their review highlights various security analytics challenges such as privacy, APT detection, cryptography, huge amount of dataset, data provenance, security visualization, and lack of skilled personnel.

The work that relates more closely to our work is that of Ricard et al. [21], which reports the challenges faced by IDS for analysing a large amount of heterogeneous data. The identified challenges include complex ML, lack of evaluation datasets, and feature selection. They recommend the correlation of security events, the collection of data from diverse sources, and incorporation of big data technologies for efficient intrusion detection. To the best of our knowledge, our work is the first SLR focussed on the architecture and architecturally significant requirements (quality attributes) of BDCA systems.

2.8 Threats to Validity

Whilst we have designed and conducted this SLR by following the guidelines of [49], there are certain threats to the validity of our findings that need to be considered while examining the findings. Based on the classification of threats to the validity reported in [149], we identified the following types of threats to the validity of the findings reported in this chapter.

Internal Validity: This type of validity refers to the threats caused by the conditions in which the study is conducted i.e., whether or not those conditions impact the findings of the study. Our SLR has the potential of missing some relevant papers. We identified and selected the relevant papers by searching the six digital repositories, which are considered the most relevant sources of literature on computer science and software engineering [49]. We did not impose any restrictions on the publication date of the papers. To ensure the reliability of the search string, we tested the search string through pilot searches and iteratively improved it until the searches returned the papers that we already knew to be relevant. We reviewed the references of the primary studies obtained via the automatic searches (i.e., snowballing) [72]. The selection of the papers was based on the inclusion and exclusion criteria

(reported in Section 2.2.3), the selection process could have been influenced by the researchers' subjective judgement. To address this issue, the reasons for inclusion and exclusion of the papers were recorded and reviewed by the authors. We also tried to reduce the potential misinterpretations to a minimum by following a multi steps selection process.

External Validity: This type of validity refers to the generalizability of the findings from the study. There can be the potential impact of a researcher's bias in extracting the data from the reviewed papers. We designed a data extraction form (Table 2.4) for ensuring consistency in data extraction. As mentioned in Section 2.2, we employed quantitative and qualitative analysis techniques for analysing the extracted data. There is a possibility of some bias in the interpretation of the extracted data. To mitigate the researchers' bias in data interpretation, wherever possible we explored the antecedent and subsequent publications of the authors of the reviewed papers, websites of the tools based on the papers, and blogs of the authors of the reviewed papers.

2.9 Chapter Summary

Motivated by the growing significance of big data analytics for cyber security, we decided to systematically gather and rigorously analyse and synthesize the literature on architectural tactics used for designing BDCA systems in this chapter. Based on a Systematic Review of 74 relevant papers, we have identified and explained 12 quality attributes considered important for BDCA systems. We have also identified and codified 17 architectural tactics for supporting the required qualities (i.e., response time, accuracy, scalability, reliability, security, and usability) in BDCA systems.

As mentioned in Section 2.3.1 the search for papers reported in Figure 2.2 was performed on in May 2017. To determine the research trend on the topic of big data cyber security analytics over the last 3-4 years, we ran our search string (Figure 2.1) on the six datasets (Table 2.2). The number of papers returned from each database for the period of June 2017 to April 2020 are presented in Table 2.8. Table 2.8 also compares the papers returned by the search string in the two periods i.e., 2009 to May 2019 and June 2017 to April 2020. In around 11 years, the total papers returned by our search query are 21,131. Among these, 12,117 (67.8%) are in the period of 2009 – May 2017 and 9,014 (32.2%) are in the period of June 2017 – April 2020. This shows that there is a substantial increase in the number of papers published in the period of June 2017 – April 2020 as compared to the period of 2009 – May 2017. This further reiterates our analysis presented in Section 2.3.1 that the role of big data technologies in cyber security domain will rise with the passage of time. This is because the volume and velocity of digital data is increasing with the passage of time due to novel technological paradigms such internet of things, fog, and edge computing. The traditional technologies (e.g., data warehouse) are not able to collect, store, and analyse such large volume of data for security analysis. Therefore, industry and academia are adopting and investigating the use of big data technologies for cyber security.

This chapter promises several potential benefits both for researchers and practitioners. For researchers, this chapter has identified a number of areas for future research. The demographic findings are of the

Table 2. 7 Number of papers returned by our search query (Figure 2.4) in the two time periods.

Source	2009- May 2017	June 2017- April 2020	Total
IEEE Xplore	897 (72.5%)	339 (27.4%)	1236
Scopus	1369 (42.1%)	1886 (57.9%)	3255
ScienceDirect	745 (96.6%)	26 (3.3%)	771
ACM	153 (56.6%)	117 (43.3%)	270
SpringerLink	8483 (56.4)	6549 (43.5%)	15032
Wiley	470 (82.8%)	97 (17.1%)	567

potential value for researchers to shape their future research directions. It is clear that the application of big data technologies in security analytics is gaining significant traction. Compared to Hadoop, Spark is becoming more popular. Given that important quality attributes such as interoperability, modifiability, adaptability, generality, stealthiness and privacy assurance lack support, we assert that researchers need to explore various options for developing such a support. Other areas for the future research include empirical evaluation of the reported architectural tactics, trade-off and dependency analysis among the tactics, and a comparative analysis among big data processing frameworks (e.g., Hadoop, Spark, and Storm) when employed in BDCA systems.

For practitioners, the identification of the most relevant quality attributes and a catalogue of architectural tactics can serve as useful design space while designing BDCA systems. Given that the elicitation of non-functional requirements (e.g., privacy, adaptability, and scalability) is a challenging task, practitioners can benefit from the identified quality attributes supported by qualitative and quantitative reasoning to establish non-functional requirements for BDCA systems. The work presented in this chapter is a first step towards guiding software architects and software engineers to efficiently architect BDCA systems. For concretizing the body of knowledge and materializing practitioners' trust through empirical evidence, we plan to establish an experimental setup of distributed computing complemented by big data technologies for rigorous empirical evaluation of the codified architectural tactics. The experimental design will aim to investigate the trade-offs and dependencies among architectural tactics. Any future effort on this topic should also include an effort to develop a body of knowledge on the strengths and weakness of big data technologies for cyber security analytics. Such a body of knowledge will help practitioners to select suitable technologies according to their specific requirements.

Chapter 3

Quantifying the Impact of Architectural Tactics for Big Data Cyber Security Analytics Systems

Related publication:

This chapter is based on our paper published in *International Conference on Parallel and Distributed Computing, Applications, and Technologies (PDCAT 2019)* "Quantifying the Impact of Design Strategies for Big Data Cyber Security Analytics: An Empirical Investigation" [6].

In Chapter 2, we observed that the state-of-the-art uses various architectural tactics (e.g., feature selection and alert ranking) to help BDCA systems to achieve the desired levels of accuracy and response time. However, the use of these tactics in the state-of-the-art is not consistent, which exposes a lack of consensus on "when to use (and not to use) these architectural tactics?" In this chapter, we follow a systematic experimentation framework to quantify the impact of four architectural tactics on the accuracy and response time with respect to three contextual factors, i.e., security data, machine learning model employed in a system, and a system's execution mode. We performed experiments on a Hadoop-based BDCA system using four security datasets, five machine learning models, and three execution modes. Our findings lead us to formulate a set of design guidelines that will help researchers and practitioners to decide when to use (and not to use) the architectural tactics.

3.1 Introduction

In Chapter 2, we reviewed 74 state-of-the-art BDCA systems to identify the architectural tactics used for achieving various quality attributes including the three most important quality attributes, i.e., accuracy, response time, and scalability. In this chapter, we only focus on accuracy and response time while scalability will be addressed in Chapter 7. Chapter 2 revealed Removal of Duplicates, Feature Selection, Signature-based Detection, and Alert Ranking as the architectural tactics for achieving accuracy and response time. However, the use of these tactics is not consistent throughout the studied BDCA systems.

Table 3.1 presents the papers among the 74 reviewed papers that use the four architectural tactics. It is evident from Table 3.1 that out of 74 papers only two papers use Removal of Duplicates, 10 papers use Feature Selection, two papers use Signature-based Detection, and only one paper uses Alert Ranking tactic. Almost 79% of the reviewed papers do not use any of these tactics in their BDCA design. This finding stimulates a question: why these studies (and not others) use the particular architectural tactics? Although the use of some tactics (e.g., Feature Selection) is well-agreed upon in various

Table 3. 1 Papers using the four architectural tactics

Architectural Tactic	Papers Using the Architectural Tactic
Removal of Duplicates	[27, 150]
Features Selection	[25, 27, 31, 39-41, 150-153]
Signature-based Detection	[154, 155]
Alert Ranking	[155]

domains (e.g., bioinformatics), a reason for such lack of a consensus on the use of the architectural tactics in BDCA is that BDCA is a relatively new breed of software systems (introduced in 2013 in a cloud security consortium [15]). Therefore, there is a need for developing a consensus on the usage of architectural tactics in BDCA. We pose this issue as follows: "***When Removal of Duplicates, Feature Selection, Signature-based Detection, and Alert Ranking tactics should be used in the design of a BDCA system?***"

In this chapter, we follow a systematic experimentation framework [156] to answer this question by quantifying the impact of the four known architectural tactics on *accuracy* and *response time* with respect to the contextual factors. By contextual factors, we mean factors that can influence the effectiveness of architectural tactics. The factors considered in this study and highlighted as most prominent by [1] and [59] include security data, the execution mode (e.g., single machine or distributed setup), and the Machine Learning (ML) model employed in a system for anomaly detection. It is worth noting that Chapter 1 reveals several architectural tactics for achieving various quality attributes in BDCA systems. However, we have selected only four architectural tactics for investigation based on the following three criteria: (i) a tactic purports to achieve either accuracy, response time, or both (ii) a tactic must be optional i.e., a system can function without it and (iii) the impact of a tactic can be precisely traced and quantified. The papers presented in Table 3.1 only report the overall *accuracy* and *response time* of a system and do not provide a complete picture of the contributions of the architectural tactics in improving end-to-end accuracy and response time. Our study fills this gap by quantifying the impact of the studied tactics on *accuracy* and *response time*. Our findings reveal that the effectiveness of the studied architectural tactics closely relates to the contextual factors. Therefore, the tactics should be used in accordance with the contextual factors. Our findings lead us to propose guidelines for making decisions about whether or not the architectural tactics should be used in their BDCA system's design.

Contributions: In this chapter, we make the following contributions.

- Report the systematic design of our experimentation approach that can be used for a similar investigation of other architectural tactics (Section 3.3 and Section 3.4)
- Quantify the impact of the studied architectural tactics on accuracy and response time that leads us to formulate guidelines for designing BDCA systems (Section 3.5)

Chapter Organization: Section 3.2 briefly describes the four architectural tactics. Section 3.3 reports experiment planning followed by experiment execution in Section 3.4. Section 3.5 presents the results. Section 3.6 discusses the results. Section 3.7 outlines the threats to the validity of the findings. Section 3.8 reports related work and finally Section 3.9 concludes the chapter.

3.2 The Four Architectural Tactics

Whilst Chapter 2 thoroughly discusses the four architectural tactics, i.e., Removal of Duplicates (Section 2.5.1.2), Feature Selection (Section 2.5.1.3), Signature-based Detection (Section 2.5.2.2), and

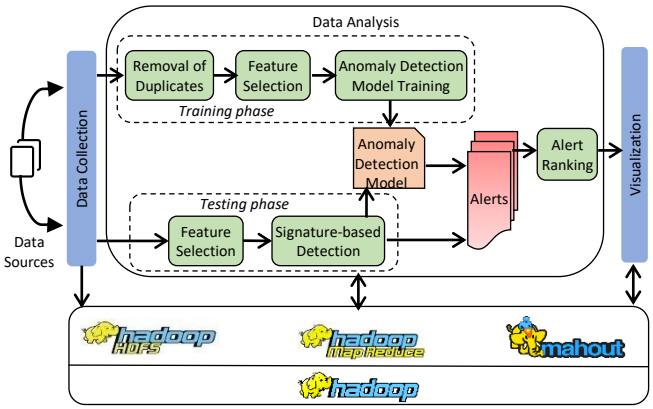


Figure 3. 1 Illustration of the four tactics in a BDCA system

Alert Ranking (Section 2.5.6.1), we briefly describe the use of each tactic in this chapter to contextualize our experimentation approach. Figure 3.1 illustrates the use of the tactics in a BDCA system's design.

3.2.1 Removal of duplicates

This tactic removes duplicate records from the collected security event data. This tactic improves response time by reducing the size of data. The *data collection* component collects security event data from different sources such as packet data, NetFlow data, and IDS logs. The selection of the data sources varies from organization to organization. After collection, the data is forwarded to the *removal of duplicates* component as shown in Figure 3.1, which processes data in a parallel fashion to remove duplicate records. This tactic is only applied in the training phase because the size of the training data is quite large that can severely hinder attack detection [150].

3.2.2 Feature selection

This tactic selects and extracts the selected features from the data. The incorporation of this tactic is motivated as each record in the security data consists of several features such as 41 features in the KDD dataset [55] and 81 features in the CICIDS2017 dataset [58]. All of these features are not pertinent to attack detection. Hence, it is essential to select features that significantly contribute to attack detection. As shown in Figure 3.1, the *feature selection* component selects the most relevant features during the training phase. In the testing phase, the already selected features (in the training phase) are selected from the data and forwarded to the subsequent components for further processing.

3.2.3 Signature-based detection

This tactic integrates a signature-based detection model with an anomaly-based detection model for detecting zero-day attacks and reducing the false positive rate. In isolation, a signature-based model, which detects attacks based on known attack patterns, is very effective in detecting known attacks but fails to detect unknown attacks (zero-day attacks) [59, 154]. An anomaly-based model, which detects attacks based on deviation from the learned normal behaviour, can detect unknown attacks, however, generates many false alarms [59, 154]. This tactic composes two models into a hybrid model to exploit the advantages of both models. As shown in Figure 3.1, the security event data is analyzed by the *signature-based detection* module and if a match is found with the attack patterns, an alert is directly generated. Having processed by the signature-based module, the data is fed into the *anomaly detection* module that analyzes the data for detecting attacks (primarily unknown attacks) that went undetected

through the signature-based module and flags alerts when a deviation from the normal behaviour is detected.

3.2.4 Alert ranking

This tactic ranks the alerts generated by a BDCA system based on the levels of severity of alerts. The incorporation of this tactic is encouraged for two reasons – (1) around 90% of alerts are not significantly dangerous [59] (2) the more dangerous an alert is, the more challenging it is to respond to it and mitigate its effects [157]. If a security operator has to respond to alerts based on the sequence of their generation, it is quite likely that the 10% alerts, which are dangerous, will be dealt with quite late. This tactic enables a security operator to prioritize response to the more dangerous alerts. The generated alerts are in a random sequence, which are ranked by the *alert ranking* component based on their dangerousness as shown in Figure 3.1.

3.3 Experiment Planning

To generate reliable and replicable results, we designed our experimental investigation according to the guidelines for experimentation in software engineering provided by Basilli et al. [156]. Accordingly, our study process consisted of (A) variable selection (B) subject selection and (C) instrumentation setup.

3.3.1 Variable selection

Two types of variables are considered in our study – Dependent and Independent. *Dependent variables* are the ones under observation for assessing a possible impact introduced due to change in the independent variable(s). We selected Accuracy and False Positive Rate (FPR) as variables for assessing the impact on accuracy and Training Time and Prediction Time as variable for assessing the impact on response time. The metrics used for measuring the variables are shown in Table 3.2. *Independent variables* are manipulated during the experiment to observe its potential impact on the dependent variable(s). The primary independent variable in our experiment is the use of the architectural tactics. Our experimental study consists of five distinct treatments – one relating to baseline and the other four to one tactic each. The remaining independent variables (e.g., configurations and machine workload) relate to the underlying software and hardware resources. These variables are ensured to remain constant during the experiment.

Table 3. 2 Evaluation metrics and their descriptions

Metric	**Description**	**Equation**
Accuracy	Percentage of correctly detected attack and normal instances	$\text{Acc} = \left(\frac{TP + TN}{TP + FP + FN + TN}\right) * 100$
False Positive Rate (FPR)	Percentage of normal instances detected as attack instances	$FPR = \left(\frac{FP}{FP + TN}\right) * 100$
Training Time	Time taken by the system in seconds to train the model using training data	
Prediction Time	Time taken by the system in seconds to detect attacks in the entire testing data	
True Positive (TP) = No. of correctly detected attack instances False Positive (FP) = No. of normal instances detected as attack instances False Negative (FN) = No. of attack instances detected as normal instances True Negative (TN) = No. of correctly detected normal instances		

3.3.2 Subject selection

The subject of our experiment is a BDCA system that purports to detect cyber attacks in an environment that generates a large volume of security event data. The architecture of such a system consists of four main components as shown in Figure 3.1. *1) Data collection:* This component collects security-related data from different sources within an organization. The potential sources for such data include but not limited to the network traffic data, application data, database access logs, user activity logs, windows logs, and honeypot data. The choice of the sources from which data should be collected varies from organization to organization. In this study, we use network traffic data as elaborated in Section 3.4.1 *2) Hadoop:* Apache Hadoop [50] is a framework for processing a large volume of data on a distributed setup of computing machines. Hadoop consists of two main parts – Hadoop Distributed File System (HDFS) and MapReduce. The HDFS is the data storage part of Hadoop, which receives data from the data collection component, partitions the data into blocks, and stores them in different nodes. MapReduce provides the programming guidelines for processing the large amount of data on multiple machines in a parallel fashion. With MapReduce, entire data is viewed in the form of key-value pairs. Two functions process the data – Mapper and Reducer. The mapper function converts each input data record into key-value pair and outputs them. Next, the key-value pairs are shuffled by the underlying Hadoop framework to group together values associated with each key. Finally, the reducer function merges the values associated with each key and outputs the results. *3) Data Analysis:* This component processes data using MapReduce. The processing involves a range of strategies that takes data in raw form, cleans it, reduces data dimensionality, and finally analyses the data using ML algorithms and rule-based approaches (e.g., signature-based detection) to extract insight about malicious activities. In this study, we use removal of duplicates tactic for cleaning the data, feature selection tactic for reducing data dimensionality, and anomaly-based detection and signature-based detection for the classification of the data into benign and malicious classes *4) Visualization:* Finally, the results generated by the data analysis component are shared with the user using visualization component. In the current form, the results are visualized in the localhost facility embedded in the Hadoop framework, however, the system can be easily integrated with advanced visualization tools (e.g., a dashboard) for more user-friendly visualization of the results.

3.3.3 Instrumentation and testbed

We implemented a BDCA system with three different modes of Hadoop version 2.6.0. The execution mode of a BDCA system closely relates to the response time [50]. Therefore, it is important to investigate the impact of all three execution modes, which are described in the following.

StandAlone (SA) mode: In this mode, Hadoop uses the local file system and all the Hadoop components (e.g., NameNode, DataNode, and JobTracker) run in a single Java process. All jobs run in the form of one mapper and one reducer. This mode mimics the behaviour of sequential execution. The test machine used for running Hadoop in SA mode is an HP EliteBook 850 G5 that hosts both master node and the data nodes responsible for running the Hadoop framework. The machine is configured with 2x Intel Xeon processors (2.60 GHz and 2.70 GHz), 8 GB RAM, 128 GB hard drive, and installed with CentOS version 6.4 operating system. The system and the studied strategies have been coded in MapReduce Java programming language on top of the Hadoop framework using Eclipse IDE. The

JVM version running on the machine for executing Java programs is JVM 1.7.0. Ultraedit[11] software has been used for manual navigation through the datasets.

Pseudo-Distributed (PD) mode: This mode mimics the behaviour of a multi-node Hadoop cluster despite being on a single machine. The different daemons of Hadoop run in different JVM instances and data is stored in HDFS instead of a local file system. The jobs run in the form of multiple mappers and reducers. The machine, IDE, and JVM version used for executing experiments in PD mode are the same as for SA mode.

Fully-Distributed (FD) mode: This mode involves multiple nodes (machines) where data processing is distributed across nodes. The Hadoop master and slave services run on separate nodes. For our experiments, we configured a Hadoop cluster (version 2.6.0) on OpenStack cloud. The cluster consisted of 17 virtual machines (1 Master and 16 slaves) each running CentOS version 6.4 operating system. Each slave node is equipped with 2 GB RAM and 20 GB disk space while the master node is equipped with 8 GB RAM and 80 GB disk space.

3.4 Experiment Execution

This section describes the operational phase of our study.

3.4.1 Experimental data

We used four different security datasets (i.e., KDD [55], DARPA [56], CIDDS [57], and CICIDS2017 [58]) in our study. The datasets are briefly described in the following with their statistics presented in Table 3.3.

KDD: The KDD dataset contains 4,898,431 records/instances as training data and 311,029 records as testing data. We use the term 'record' and 'instance' interchangeably to represent a network connection. Each record consists of 41 features. The training dataset contains around 78% duplicate records. Each record is labelled as belonging to either the normal class or one of the four attack classes i.e., *Denial of Service (DoS), Probing (Probe), Remote to Local (R2L), and User to Root (U2R)*. The testing data includes attack types that are not present in the training data, which makes the evaluation more realistic.

DARPA: Similar to KDD, the records in this dataset are divided into training data and testing data. The training data consists of 2,723,496 records while the testing data consists of 863,513 records. Each record represents a network connection – consisting of 11 features. The training subset of this dataset contains around 15% duplicate records. Each record is labelled as 0 or 1, where '0' specifies a normal connection and '1' specifies an attack. The attack types present in DARPA are the same as KDD.

CIDDS: This dataset has been recently developed to address the problem of KDD and DARPA being quite old. The dataset consists of four-week NetFlow data directed towards two servers (i.e., OpenStack and External Server). We used 66% data in the dataset for training and 34% for testing the model. Our training dataset consists of 5,634,348 records and testing dataset consists of 2,817,173 records. Each record represents a network connection – consisting of 12 features. The training subset of this dataset contains around 41% duplicate records. The dataset contains four types of attacks*: pingScan*, *portScan*, *bruteForce*, and *DoS*.

[11] https://www.ultraedit.com/

Table 3. 3 Statistics of the datasets

Dataset	Connection Type	Training Data		Testing Data		Attack Types
		No. of Records	Percentage of Records	No. of Records	Percentage of Records	
KDD	Normal	972,781	19.8%	60,593	19.4%	Denial of Service, Remote to Local, Probing, and User to Root
	Attack	3,925,65	80.14%	250,436	80.5%	
DARPA	Normal	1,145,94	42.07%	658,797	43.3%	Denial of Service, Remote to Local, Probing, and User to Root
	Attack	1,577,55	57.92%	863,513	56.7%	
CIDDS	Normal	4,706,99	83.54%	2,303,89	81.78%	Port Scan, Ping Scan, Brute Force, Denial of Service
	Attack	927,349	16.46%	513,275	18.22%	
CICIDS2017	Normal	1,109,93	84.6%	555,005	84.6%	Brute Force, Distributed DoS, DoS, Botnet, Web Attack, Heart Bleed, and Infiltration attack
	Attack	201,889	15.4%	100,905	15.4%	

CICIDS2017: This is also a recently developed dataset, which contains a variety of state-of-the-art attacks. The dataset consists of five days of network traffic directed towards a network consisting of three servers, a firewall, a switch, and 10 PCs. Like CIDDS, we used 66% data for training and 34% for testing. Our training dataset consists of 1,311,822 records and test dataset consists of 655,910 records. Each record consists of 80 features and only 0.2% records in the training dataset are duplicate. This dataset contains six types of attacks: *bruteForce*, *heartBleed*, *botNet*, *DoS*, *Distributed DoS*, *webAttack*, and *infiltration attack*.

3.4.2 Experimental treatments

We categorized our experimentation process into five treatments, which are described in the following.

Treatment 1: Baseline: In this case, the system does not use any of the four tactics. The baseline is the controlled situation used to compare our system's accuracy and response time with the four experimental cases. In this case, a system only leverages ML algorithm for anomaly detection (as shown in Figure 3.1) without any pre or post-processing of data. The algorithms evaluated in this study include K-means (KM), XGBoost (XGB), Naïve Bayes (NB), Support Vector Machine (SVM), and Random Forest (RF). The algorithms are implemented using Mahout [52], which is an ML library for distributed ML algorithms.

Treatment 2: Removal of Duplicates: We consider records duplicate when all the features of the two instances are exactly the same. The existence of these exactly alike records makes an ML model biased towards more frequent records, hence, preventing it from learning less frequent behaviours pertaining to more dangerous attacks such as R2L and U2R in KDD [55]. For implementation, we used the *NullWritable* class available in the *org.apache.hadoop.io* library. We input the training data to the mapper function, which outputs each line as the key and *NullWritable* as the value. The Hadoop framework shuffles the generated key-value pairs that automatically removes the duplicated records as distinct keys are supposed to be forwarded to the reducer function. The reducer takes the key-value pairs and writes the received keys (i.e., distinct records) and *NullWritable* to the output using the context object.

Treatment 3: Feature Selection: We employ Information Gain (InfoGain) approach proposed in [142] to select the most pertinent features. We select *InfoGain* as a representative technique due to its widespread usage in BDCA [1]. For *InfoGain*, we compute the entropies for each feature. Features having high relevance to data classification will have low entropy value and consequently high *InfoGain*. Our implementation selects 12, 6, 8, and 16 features from 41, 9, 12, and 81 features available

Table 3. 4 Features selected for each dataset

Dataset	Selected Features
KDD	Duration, Protocol, Service, Flag, src_bytes, dst_bytes, num_root, dst_host, srv_count, dst_host_Same_src_port_Rate, dst_host_srv_diff_host_rate, dst_host_serror_rate
DARPA	Duration, Service, Source port, Destination port, Source IP, Destination IP
CIDDS	Duration, Protocol/Service, Source IP, Source Port, Destination IP, Destination Port, Packets, Bytes
CICIDS2017	Total Fwd Packets, Total Backward Packets, Source IP, Source Port, Destination IP, Destination Port, Subflow F.Bytes, Total Len F.Packets, B.Packet Len Std, B.Packet Len Std, B.Packet Len Std, B.Packet Len Std, Subflow F.Bytes, B.Packet Len Std, Flow Duration, Total Len F.Packets,

in KDD, DARPA, CIDDS, and CICIDS2017 datasets respectively. These selected features for each dataset are shown in Table 3.4.

Treatment 4: Signature-based Detection: For implementing this strategy, we used training dataset to build two separate models – a signature-based detection model using C4.5 decision tree and an anomaly-based detection model using algorithms mentioned in Treatment 1. For building the signature-based model, we followed the approach adopted in [158] – selecting C4.5 for building the signature-based detection model based on its ability to generate signatures that can accurately detect known attacks. Some of the sample signatures (for detecting *smurf*, *neptune*, and *satan* attacks) embodied in the signature-based model are shown in Figure 3.2. Once the models are developed, the testing dataset is first passed through the signature-based model. Based on the learned signatures, this model classifies the connections in the dataset as being an attack or normal. The connections identified as attacks are concluded as confirmed attacks while the connections classified as normal are passed through the anomaly-based detection model. An attack is blocked, otherwise, the connection is allowed for the network.

Treatment 5: Alert Ranking: Our implemented system's alerts are ranked using the ranking model presented in [157]. According to this model, the alerts that cause unexpected variations in the network are more dangerous, thereby, we rank alerts based on the number of bytes pertaining to each alert. The more the number of bytes, the more dangerous is an alert. At the implementation level, the mapper function extracts *src_bytes* and *dst_bytes* from each alert and sum them to get the number of bytes embodied in the alert. The processing framework shuffles and sorts the intermediate results based on the generated keys. Followed by this, the reducer function writes the received key-value pair to the output. Due to the absence of required features, the strategy is not applied on DARPA dataset.

3.4.3 Data collection and data analysis

We used CentOS virtual machine for running all the experiments. The java codes for all treatments were first written, compiled, and debugged (where necessary) for all treatments in Eclipse IDE. After compilation, the compiled code was exported as a Jar file. Since Hadoop jobs depend upon several libraries, we made sure that all the dependencies are exported in the Jar file. The exported Jar file was

```
IF wrong_fragment <= 0.500 AND serror_rate>322.50 AND src_bytes>518 THEN attack = smurf

IF wrong_fragment<=0.500 AND serror_rate<=322.50 AND land<=0.500 AND srv_serror_rate>0.925 AND
dst_host_rerror_rate>0.955 THEN attack = neptune

IF wrong_fragment <=6.500 AND serror_rate<=322.50 AND land<=0.500 AND srv_serror_rate<=0.325 AND
src_bytes>6.500 AND srv_rerror_rate>0.110 THEN attack = satan
```

Figure 3. 2 Sample signatures for detecting smurf, neptune, and satan attacks

placed on the master node in the already deployed OpenStack cluster. The data (e.g., KDD dataset) associated with the Jar file was uploaded to the cluster. Before running the job, the connectivity between the master node and workers nodes in the cluster used to be checked. We also used to check the health of worker nodes to make sure that the nodes do not exhaust or crash during the execution. After the pre-checks, we used to submit each job to the cluster through CentOS terminal for execution and monitor the operational logs on the terminal for any errors. In case of any errors relating to the code, we used to fix and again export the Jar file. For each of the treatments, we executed the experiment 10 times. Each treatment generated a text file containing the generated alerts and the logged data about job executions. For calculating the evaluation metrics, we used the equations shown in Table 3.2. For calculating training time and prediction time, we leverage the logged information, which contains the job's starting time, job's ending time, mapper time, shuffling time, and reducer time. In Section 3.5, the results are reported as an average of the values collected in 10 executions.

3.5 Results

3.5.1 When to use Removal of Duplicates tactic?

- *Do not use Removal of Duplicates tactic for datasets (i) with less than 10% duplicate instances and (ii) with duplication distributed among multiple classes (e.g., DoS, U2R).*
- *Use Removal of Duplicates tactic with K-means and XGBoost.*
- *Use Removal of Duplicates tactic irrespective of the execution mode.*

Figure 3.3 shows the accuracy and False Positive Rate (FPR) and Table 3.5 presents the training and prediction time achieved with and without Removal of Duplicates tactic. The impact of the strategy on accuracy and training time is aligned with the percentage of duplicate instances in each dataset. For example, KDD dataset having 78% duplication benefits the most (i.e., 3.61% accuracy and 43.1% improvement in training time) and CICIDS2017 with merely 0.2% duplication benefits the least (i.e., 0.04% accuracy and 0.72% improvement in training time) from the tactic. Therefore, we assert that the tactic should be used when the training dataset has a large proportion of duplicate instances. Otherwise, passing the data (with few duplicates instances) through the removal of duplicates filter will incur a processing cost that will cancel the benefit expected through the removal of duplicate instances. Since the tactic is more effective for two datasets (i.e., KDD and DARPA) that have more than 10% duplication and less effective for two datasets (i.e., CIDDS and CICIDS2017) that have less than 10% duplication, we recommend that the Removal of Duplicates tactics should not be used for dataset with less than 10% duplication. Furthermore, the impact on accuracy is higher when the duplication belongs to a particular class such as DoS class in DARPA dataset. This is because a concentration of the duplicate instances in a particular class makes the ML model biased towards either positive or negative samples for the same class and hence makes the model unable to classify the instances accurately for the particular class. Therefore, we recommend that the Removal of Duplicates tactic should not be used for data that has duplication belonging to a specific class. Among the used classifiers, K-means and XGBoost yield the largest accuracy improvement– an improvement of around 3.23% on average. This is followed by Random Forest (1.67%) and SVM (1.19%). The impact for Naïve Bayes is negligible (i.e., 0.05%). XGBoost benefitted the most from the tactic in terms of training time – a reduction of 24.86%. Given the impact of the tactic on the accuracy and training time for various algorithms, we suggest that the tactic should be definitely used with K-means and XGBoost algorithm. In all three execution modes, the tactic reduces training time significantly i.e., 23.01% in PD mode,

21.75% in SA mode, and 17.72% in FD mode. Hence, the strategy should be used irrespective of the execution mode.

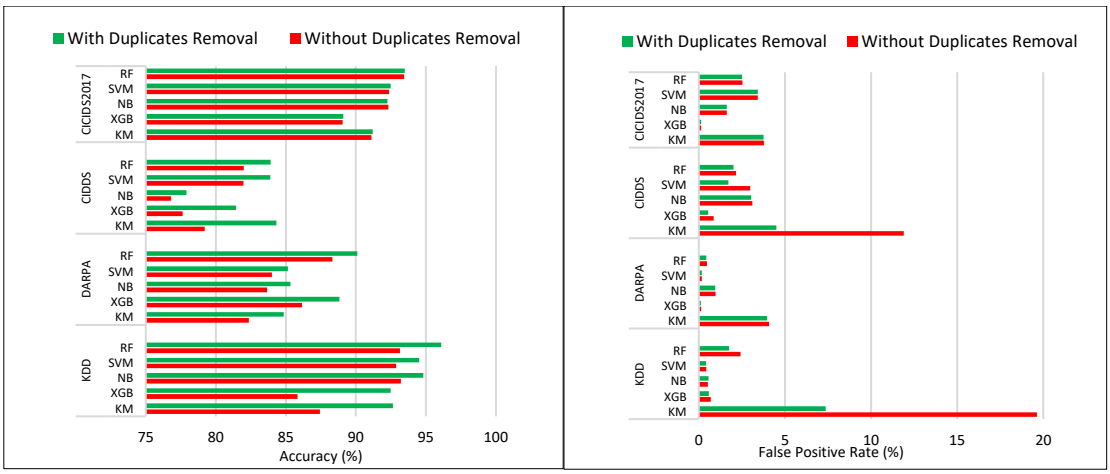


Figure 3. 3 Accuracy and false positive rate with and without Removal of Duplicates

Table 3. 5 Training time and prediction time with and without Removal of Duplicates

		Training Time (sec)						Prediction Time (sec)					
		Pseudo-distributed		Standalone		Fully-distributed		Pseudo-distributed		Standalone		Fully-distributed	
		Without Duplicate Removal	With Duplicates Removal	Without Duplicate Removal	With Duplicates Removal	Without Duplicate Removal	With Duplicates Removal	Without Duplicate Removal	With Duplicates Removal	Without Duplicate Removal	With Duplicates Removal	Without Duplicate Removal	With Duplicates Removal
KDD	K-means	2090	1101	1566	807.3	1220	721.8	19.6	19.5	13.6	13.62	11.3	11.29
	XGBoost	4242	1964	3654	2228	3005	1566	7.56	7.56	6.12	6.12	6.91	6.9
	Naïve Bayes	370.2	255.4	246.3	154.9	201.6	141.3	28.4	28.3	22.34	22.32	20.45	20.43
	SVM	588.9	367.1	401.2	219.8	341.1	191.4	22.14	22.14	18.64	18.65	15.67	15.67
	Random Forest	425.6	221.4	310.4	154.3	264.7	142	19.22	19.2	17.1	17.12	13.21	13.22
DARPA	K-means	1145	773	772.2	616.1	532.6	494.8	34.52	34.52	22.97	22.98	18.1	18
	XGBoost	8020	4138	6515	4133	3314	3117	27.33	27.12	20.6	20.4	18.1	18.05
	Naïve Bayes	1290	1011	876.6	667.4	643.2	616.8	63	63	54.4	54.3	48.22	48.21
	SVM	1860	1420	1211	998.2	987.2	668.9	45.2	45.1	34.62	34.61	29.51	29.48
	Random Forest	1501	921.7	1002	778.4	801.7	684.2	34.66	34.66	28.11	28.09	20.88	20.8
CIDDS	K-means	1776	1444	1278	1011	981	827	42.4	42.3	31.3	31.3	29.6	29.5
	XGBoost	9118	6678	7761	5443	4416	3991	31	29.8	20.7	20.6	17.11	17.1
	Naïve Bayes	2133	1910	1771	1612	1291	1091	48	47.8	34.3	34.2	15.64	15.61
	SVM	2838	2621	2214	2011	1601	1400	36	35.5	22	21.5	12.98	12.91
	Random Forest	2438	2178	2168	1708	1491	1109	32.77	32.72	22.12	22.05	11.19	11.17
CICIDS 2017	K-means	1508	1497	1332	1301	1099	1091	23.8	23.8	16.4	16.4	12.4	12.3
	XGBoost	6233	6225	4901	4889	3872	3865	12.56	12.54	9.54	9.52	6.88	6.81
	Naïve Bayes	656.5	651	444.3	437	298	296	58.62	58.61	31.2	31.22	19.34	19.31
	SVM	778.2	765.9	571	571	339.6	337	44.67	44.61	36.32	36.29	24.44	24.41
	Random Forest	701.8	699	497	496.3	312	310.6	38.81	38.79	32.77	32.7	20.2	20.1

3.5.2 When to use Feature Selection tactic?

- *Use Feature Selection for datasets (i) with 10 or above number of features and (ii) with a larger proportion of numerical features.*
- *Use Features Selection with all ML algorithms*
- *Use Feature Selection strategy in all execution modes but is most recommendable in standalone mode*

Figure 3.4 shows the accuracy and FPR and Table 3.6 presents the training and prediction time with and without Feature Selection. The impact of the tactic is aligned with the number of features in each dataset. CICIDS2017 with 81 features benefits the most in terms of accuracy (3.09%), training time (27.7%), and prediction time (12.9%). It is worth noting that the accuracy for DARPA with only nine features is reduced by 0.27%. Based on our experimental results, the accuracy and prediction time is improved for three datasets (i.e., KDD, CIDDS, and CICIDS2017) that have more than 10 features and negatively impacted for one dataset (i.e., DARPA) that has less than 10 features. Therefore, we suggest that the strategy be used for datasets with at least 10 or above number of features. This is because passing a dataset, that has less than 10 features, through a feature selection filter will only consume the filtering time without a significant gain in accuracy and response time in the subsequent step. The tactic is more effective for datasets with a larger proportion of numerical features. For instance, CIDDS have almost the same number of features as DARPA but a greater number of numerical features benefits more from the tactic. Hence, we recommend to use Feature Selection tactic especially for data with a large proportion of numerical features. All ML algorithms benefit from this strategy with XGBoost benefiting the most in terms of accuracy (2.18%) and prediction time (8.32%) and random forest benefiting the most in terms of training time (18%). Considering that all ML algorithms benefit from the use of tactic in one form or another with no evident negative impact in relation to the ML algorithms, we recommend that the tactic be used with all ML algorithms. The average reduction in training and prediction time with respect to the execution modes is 10.43% (SA), 9.86% (FD), and 7.62% (PD). The reduction in training time and prediction time is not in accordance with the reduction in data size. For example, reducing the number of features from 41 to 12 in KDD reduces the size of the dataset by 68.06%. However, the reduction in training time for KDD is on average 20.6%. This can be attributed to the Hadoop model, which requires separate read and write for each record in PD and FD mode. This is why the tactic is most effective in SA mode, which does not require separate read and write. Since the tactic significantly reduces the prediction time in all the execution modes, we recommend its use for all execution modes. However, the reduction is quite high in standalone mode due to its operational nature, therefore, it is most recommended to use the Feature Selection tactic in the standalone mode.

Figure 3. 4 Accuracy and false positive rate with and without Feature Selection

Table 3. 6 Training time and prediction time with and without Feature Selection

		Training Time (sec)						Prediction Time (sec)					
		Pseudo-distributed		Standalone		Fully-distributed		Pseudo-distributed		Standalone		Fully-distributed	
		Without Feature Selection	With Feature Selection	Without Feature Selection	With Feature Selection	Without Feature Selection	With Feature Selection	Without Feature Selection	With Feature Selection	Without Feature Selection	With Feature Selection	Without Feature Selection	With Feature Selection
KDD	K-means	632	538.2	497	378	348	304	19.6	19.4	13.6	13.3	11.3	11
	XGBoost	1480	1133	979	781	788	558	7.56	7.42	6.12	5.84	6.91	6.09
	Naïve Bayes	370.2	323	246.3	197.3	201.6	181.4	28.4	27.8	22.34	21.94	20.45	20.01
	SVM	588.9	457	401.2	301.5	341.1	278	22.14	22	18.64	18.51	15.67	15.48
	Random Forest	425.6	329	310.4	238	264.7	198	19.22	18.84	17.1	16.12	13.21	12.41
DARPA	K-means	1145	1105	772.2	732.5	532.6	497	34.52	34.53	22.97	22.99	18.1	18.3
	XGBoost	8020	7975	6515	6447	3314	3229	27.33	27.21	20.6	20.42	18.1	18.2
	Naïve Bayes	1290	1271	876.6	859.5	643.2	629	63	63.1	54.4	54.4	48.22	48.24
	SVM	1860	1801	1211	1119	987.2	909	45.2	45	34.62	33.98	29.51	29.48
	Random Forest	1501	1397	1002	881	801.7	701.4	34.66	34.51	28.11	28.08	20.88	20.78
CIDDS	K-means	1776	1706	1278	1198	981	887	42.4	42.2	31.3	30.9	29.6	29.4
	XGBoost	9118	9060	7761	7685	4416	4337	31	29.21	20.7	19.3	17.11	16.84
	Naïve Bayes	2133	2087	1771	1654	1291	1168	48	48	34.3	34.3	15.64	15.63
	SVM	2838	2746	2214	2167	1601	1509	36	35.1	22	21	12.98	12.02
	Random Forest	2438	2347	2168	2079	1491	1387	32.77	32.09	22.12	21.97	11.19	11
CICIDS 2017	K-means	1508	1097	1332	964	1181	797	23.8	19.7	16.4	13.4	12.4	10.9
	XGBoost	6233	5901	4901	3891	3872	3259	12.56	9.56	9.54	8.31	6.88	5.54
	Naïve Bayes	656.5	499.5	444.3	301.7	298	188	58.62	56.31	31.2	29.1	19.34	17.31
	SVM	778.2	567.7	571	411	339.6	201.5	44.67	41.2	36.32	33.17	24.44	21.5
	Random Forest	701.8	518.3	497	321	312	197	38.81	34.5	32.77	28.5	20.2	19.1

3.5.3 When to use Signature-based Detection tactic?

- *Do not use Signature-based Detection strategy if (i) the testing data contains attacks that are not present in training data and (ii) the dataset has a large proportion of categorical features*
- *Use Signature-based Detection strategy if testing data contains a large proportion of known attack instances*
- *Use Signature-based Detection strategy in situations where frequent update of the ML model is not required*

Figure 3.5 shows the accuracy and FPR and Table 3.7 presents the training and prediction time with and without Signature-based Detection tactic. The average improvement in accuracy is recorded as 3.13% for DARPA, 2.84% for CIDDS, 2.23% for CICIDS2017, and 2.4% for KDD. Concerning accuracy, the gain is lower for datasets (e.g., KDD) having attack types present in the testing data but not in the

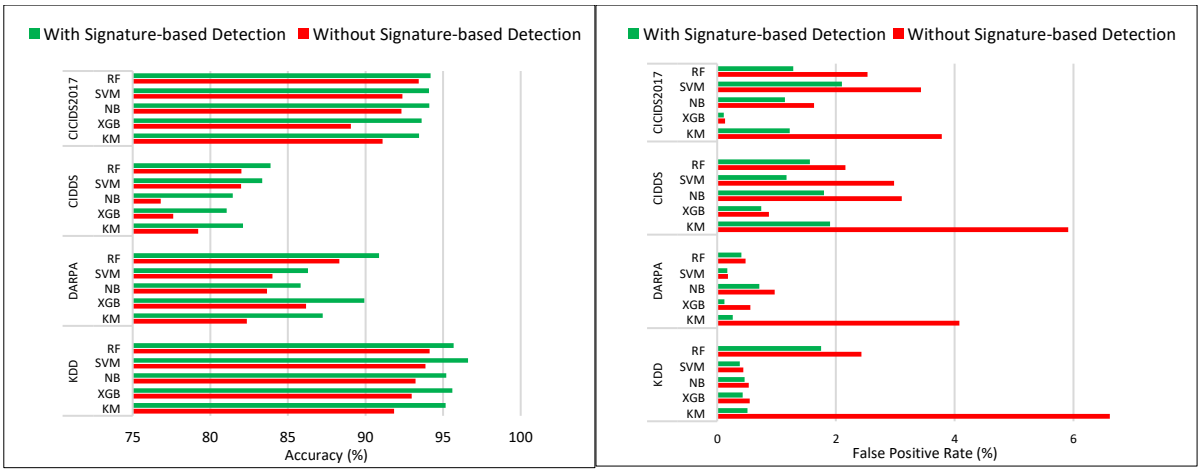


Figure 3. 5 Accuracy and false positive rate with and without Signature-based Detection

Table 3. 7 Training time and prediction time with and without Signature-based Detection

		Training Time (sec)						Prediction Time (sec)					
		Pseudo-distributed		Standalone		Fully-distributed		Pseudo-distributed		Standalone		Fully-distributed	
		Without Signature-based Detection	With Signature-based Detection	Without Signature-based Detection	With Signature-based Detection	Without Signature-based Detection	With Signature-based Detection	Without Signature-based Detection	With Signature-based Detection	Without Signature-based Detection	With Signature-based Detection	Without Signature-based Detection	With Signature-based Detection
KDD	Kmeans	632	1212	497	894	348	649	19.6	11.6	13.6	9.2	11.3	11
	XGBoost	1480	2060	979	1376	788	1089	7.56	8.54	6.12	6.54	6.91	8.5
	Naives Bayes	370.2	950.2	246.3	643.3	201.6	502.6	28.4	22.5	22.34	18.01	20.45	17.56
	SVM	588.9	1168	401.2	798.2	341.1	642.1	22.14	15.65	18.64	13.33	15.67	14.65
	Rand Forest	425.6	1005	310.4	707.4	264.7	565.7	19.22	14.6	17.1	13	13.21	13
DARPA	Kmeans	1145	2805	772.2	1979	532.6	1445	34.53	28	22.97	18.44	18.1	16.5
	XGBoost	8020	9680	6515	7722	3314	4227	27.33	32.7	20.6	19.5	18.1	15.7
	Naives Bayes	1290	2950	876.6	2083	643.2	1556	63	54	54.4	46.8	48.22	39.55
	SVM	1860	3520	1211	2418	987.2	1900	45.2	37	34.62	28.5	29.51	24.6
	Rand Forest	1501	3161	1002	2209	801.7	1714	34.66	29	28.11	23.4	20.88	18.1
CIDDS	Kmeans	1776	4421	1278	3559	981	2826	42.4	37.5	31.3	28.5	29.6	25.7
	XGBoost	9118	11763	7761	10042	4416	6261	31	33	20.7	21.5	17.11	15.6
	Naives Bayes	2133	4778	1771	4052	1291	3136	48	41.5	34.3	31.5	15.64	14.3
	SVM	2838	5483	2214	4495	1601	3446	36	34	22	21	12.98	12
	Rand Forest	2438	5083	2168	4449	1491	3336	32.77	29.5	22.12	21	11.19	10.6
CICIDS 2017	Kmeans	1508	2213	1332	1845	1181	1582	23.8	20.5	16.4	14.3	12.4	11.5
	XGBoost	6233	6938	4901	5414	3872	4273	12.56	14.56	9.54	11.4	6.88	7.51
	Naives Bayes	656.5	1361	444.3	957.3	298	699	58.62	49.7	31.2	27.5	19.34	18.1
	SVM	778.2	1483	571	1084	339.6	740.6	44.67	41.55	36.32	31.6	24.44	21.5
	Rand Forest	701.8	1406	497	1010	312	713	38.81	34.5	32.77	28.44	20.2	18.6

training data. Therefore, we recommend that in operating environments where zero-day attacks are expected, the gain from this tactic may not be significant. Moreover, this tactic is more effective for datasets with a larger proportion of categorical features such as DAPRA and CIDDS. The use of the tactic reduced the prediction time by 9.67% on average. The reason for this is that signature-based detection is quite fast as compared to anomaly-based detection. As mentioned in Section 3.4.2, the data is first processed by the signature-based detector, which excludes the connections identified as attacks and forwards the remaining connections to the anomaly-based detector. Hence, the larger the proportion of attack instances in the dataset, the larger is the gain in prediction time as evident from the largest gain for KDD (13.4%) with 80.5% attack instances and smallest gain for CIDIDS2017 (5.8%) with only 15.4% attack instances. Thereby, the tactic should be used in operating environment where majority of the network traffic is expected to be malicious and minority of the traffic is benign. The signature-based detector will remove the majority of the traffic – flagged as attached and so the anomaly-based detector will have to analyse the left out benign traffic for a second phase of screening. On average, the use of the Signature Detection tactic increases the training time by 46.06%. This is because, with the incorporation of this tactic, the system essentially trains two models, i.e., a signature-based detection model (C4.5) and an anomaly-based detection model (e.g., K-means or XGBoost). Thus, the training time is calculated as the sum of the training time for both the models. Thereby, the use of the tactic is not recommended in situations where frequent update of the ML model is required.

3.5.4 When to use Alert Ranking tactic?

- *Use Alert Ranking strategy for datasets with large variance for features (e.g., source and destination bytes) used in alert ranking*
- *Do not use Alert Ranking for datasets with a large number of instances*
- *Use Alert Ranking strategy primarily in Fully distributed mode*

We qualitatively assessed the ranked alerts for each dataset to investigate whether the alerts ranked at top are more dangerous as compared to others. For KDD, we observed that the alerts ranked on top mostly belong to *xlock* and *warezmaster* attacks while alerts ranked at the bottom primarily belong to *Neptune* and *apache2* categories. From the dangerousness perspective, the *xlock* attack fools a legitimate user to reveal the password and render full access to the attacker while *warezmaster* attack uploads illegal software to server for exfiltrating data and disrupting the server's operations [159]. The *Neptune* and *Apache2* are DoS attacks where the attacking system keeps sending connection requests to the affected system. In some cases, the affected system may crash, exhausts memory or becomes inoperative. However, in most cases, there is a connection timeout, which means the victim system will eventually recover from the attack [159]. These attacks can be considered less severe as compared to attacks like *xlock* and *warezmaster*. For CIDDS, the alerts ranked on top mostly belong to *bruteForce* attack while alerts ranked at bottom belong to *portScan* and *DoS*. Once successfully executed, *bruteForce* attacks are considerably more dangerous as it allows full access to the victim machine. On the other hand, *portScan* attacks only exfiltrate information about services running on the machine. A decent firewall setup usually blocks attacks of such nature. We observe the same trend for CICIDS2017, where attacks ranked at the top (i.e., *bruteForce* and *Botnet*) are considered more dangerous as compared to alerts ranked at the bottom (i.e., *DoS* and *SQL injection*). Our qualitative analysis suggests that such a ranking of alerts increases usability by enabling a user to prioritize response to more severe attacks. Such enhanced usability adds to the accuracy of a system to deal with severe attacks on prioritized basis. Table 3.8 presents the training and prediction time of the BDCA system with and without alert ranking. On average, alert ranking increases the prediction time by 46.2%. The dataset most impacted

Table 3. 8 Training time and prediction time with and without Alert Ranking

		Training Time (sec)						**Prediction Time (sec)**					
		Psedo-distributed		**Standalone**		**Fully-distributed**		**Pseudo-distributed**		**Standalone**		**Fully-distributed**	
		Without Alert Ranking	With Alert Ranking	Without Alert Ranking	With Alert Ranking	Without Alert Ranking	With Alert Ranking	Without Alert Ranking	With Alert Ranking	Without Alert Ranking	With Alert Ranking	Without Alert Ranking	With Alert Ranking
KDD	Kmeans	632	630.2	497	496	348	347	19.6	40.6	13.6	26.54	11.3	22.3
	XGBoost	1480	1484	979	978	788	788.6	7.56	24.84	6.12	18.06	6.91	15.91
	Naives Bayes	370.2	368	246.3	246	201.6	200.5	28.4	50.4	22.34	35.74	20.45	31.45
	SVM	588.9	589.5	401.2	402.5	341.1	341.1	22.14	38.14	18.64	30.04	15.67	26.67
	Rand Forest	425.6	424.3	310.4	310.5	264.7	263.8	19.22	35.22	19.1	30.5	13.21	22.21
CIDDS	Kmeans	1776	1777	1278	1279	981	984	42.4	72.7	31.3	53.6	31.6	48.8
	XGBoost	9118	9116	7761	7754	4416	4412	31	59.3	20.7	42.3	17.11	32.31
	Naives Bayes	2133	2133	1771	1776	1291	1289	48	82.3	34.3	58.6	15.64	30.84
	SVM	2838	2836	2214	2217	1601	1601	36	68.3	22	46.3	12.98	28.18
	Rand Forest	2438	2434	2168	2168	1491	1488	32.77	61.07	22.12	44.42	11.19	26.39
CICIDS 2017	Kmeans	1508	1507	1332	1331	1181	1184	23.8	48.8	16.4	32.8	12.4	25.6
	XGBoost	6233	6230	4901	4905	3872	3871	12.56	37.56	9.54	25.94	6.88	20.08
	Naives Bayes	656.5	656	444.3	444	298	295	58.62	83.62	31.2	47.6	19.34	32.54
	SVM	778.2	778.8	571	568	339.6	340.5	44.67	69.67	36.32	52.72	24.44	37.64
	Rand Forest	701.8	700.5	497	496	312	310	38.81	63.81	32.77	49.17	20.2	33.4

by this tactic is CIDDS (47.3% increase) followed by KDD (45.9% increase) and CICIDS2017 (44.4% increase). This trend follows the number of records in each dataset i.e., the larger the number of records, the larger is the impact. Hence, for datasets for with a large number of instances, the alert ranking tactic can be supplemented with additional computational power to reduce the impact on prediction time. Since the increase in prediction time is quite significant, we also assert that the tactic should be used only if the variance for source bytes and destination bytes is high. This is because if the instances in a dataset do not vary too much in terms of source and destination bytes, the ranking will not be very useful as these are the two features used for ranking. The impact of this strategy on prediction time is lowest in FD mode (i.e., 34.5%) as compared to SA and PD modes. The reason being that FD mode has the most computational power, which reduces the impact on prediction time. Therefore, it is recommended to use Alert Ranking strategy in FD mode.

3.6 Discussion

In this section, we discuss the results with respect to the three contextual factors i.e., the execution mode, the ML algorithms, and the security datasets.

3.6.1 Impact of execution mode

Our results show that the training and prediction time varies among different execution modes (i.e., PD, SA, and FD). With regards to the training time, the average difference is 20.5% from PD to SA, 31.6% from SA to FD, and 45.6% from PD to FD. Similarly, the difference in prediction time is 26.14% from PD to SA, 24.84% from SA to FD, and 44.5% from PD to FD. These differences are not in accordance with the processing capability allocated to each mode i.e., 8 GB to PD and SA each, and 32 GB to FD. From our findings, we can conclude that running Hadoop in PD mode on a single machine is far slower (23.5% in our case) as compared to sequential execution (i.e., SA). Furthermore, the impact of distributed processing is far more effective for larger datasets as evident from the significant difference of impact observed for larger datasets (i.e., CIDDS) as compared to the impact for the smaller datasets (i.e., KDD) when we migrate from SA to FD. Hence, we suggest to favor FD mode over SA mode only if the size of data is considerably large.

SA mode is 23.5% faster than PD on a single machine. FA mode is not effective (as compared to SA mode) for processing small size data.

3.6.2 ML algorithm selection

It is evident from the results that ML algorithms play a vital role in the fast and accurate detection of attacks. However, the selection of an algorithm is a critical concern. This is because an algorithm working best for one quality of data does not work as efficiently for another quality of data. For example, XGBoost generates more accurate results for DARPA while SVM does so for KDD. Moreover, our results show that an algorithm impacts different qualities differently. For example, the prediction time for XGBoost is quite fast but it is not as accurate as K-means. This means an algorithm delivering fast results may not necessarily generate accurate results too. Based on these findings, we suggest an adaptive approach for the selection and application of ML algorithms. Accordingly, a BDCA system should be equipped with multiple algorithms from which the most suitable can be selected according to changes in data quality and speed.

The effectiveness of ML algorithms significantly varies across security datasets. A trade-off between accuracy and response time is critical during ML algorithm selection

3.6.3 Role of dataset

We also investigate the role of a dataset in the accuracy and response time of our BDCA system. Since the used datasets both with a small and a large number of features require individual read and write for each instance (as per Hadoop framework), the difference in processing time for instance with small or large number of features is negligible as revealed in Section 3.5.2. Therefore, we recommend that feature expansion techniques [160] be used, especially for datasets with small number of features, in BDCA to achieve higher accuracy without a significant impact on response time. As expected, our results show that the larger the number of instances in the datasets, the larger is the response time. We also reflected on whether there is a relationship between the number of instances and system's accuracy. Our results nullify any such relationship. For instance, CIDDS dataset with 5,634,348 instances in the training data achieves a mean accuracy of 79.5% while CICIDS2017 with merely 1,311,822 instances achieved a mean accuracy of 91.6%. Hence, we assert that it is the quality of instances and not the quantity that determines the accuracy of a BDCA system.

Feature expansion should be used for datasets with a small number (e.g., 10) of features. The accuracy of a BDCA system does not relate with the number of instances in the dataset

3.7 Threats to validity

External Validity: The specific implementation of the studied tactics, the variety of ML algorithms, and the security datasets poses a threat to the generalization of our findings. To mitigate these threats, we used the most widely adopted implementation techniques, ML algorithms, and security datasets used in the BDCA domain. Hence, we assert that our findings are generalized within the scope of the BDCA domain. However, we do believe that the pool of algorithms, datasets, and implementation options can be extended in the future to get results that are generalizable to domains beyond BDCA**.** ***Internal Validity:*** As mentioned in Section 3.3.1 that the environmental variables such as network bandwidth, load on the computing node, and health of the worker nodes poses a threat to our findings. However, we ensured that these variables remain constant during experiment execution. We also executed each experiment 10 times to remove any fluctuations caused by the variation in environmental variables.

3.8 Related work

Table 3.9 presents the related studies on BDCA that use at least one of the tactics along with ML algorithms, datasets, and execution mode used in the study. Table 3.9 also includes a couple of BDCA studies published after our SLR was conducted. The related studies use different datasets, ML algorithms, and execution modes, therefore, an apple to apple comparison with our results is not possible. Although the studies use the strategies, none of the studies has quantified and reported the individual impact of the design strategies. Moreover, the studies are evaluated with a few ML algorithms, either one or two datasets, and a single execution mode. Our study, which is the first of its kind that follows a systematic experimentation framework, has tried to fill these gaps by quantifying the individual impact of the strategies with respect to five ML algorithms, four datasets, and three execution modes. Our findings lead us to formulate design guidelines for using the studied tactics, which are missing in the related studies.

Table 3. 9 Algorithms, datasets, and execution modes used in the related studies

Study Reference	Design Strategy				Algorithm	Dataset	Exection mode
	Removal of Duplicates	Feature Selection	Signature-based Detection	Alert Ranking			
[150]	✓	✓			K-means, SVM, KNN	KDD, CDMC2012	FD
[27]	✓	✓			Rule-based attack detection	Simulation	PD
[161]	✓				K-means		
[41]		✓			Naïve Bayes, Random Forest, SVM, J48	KDD	PD
[39]		✓			SVM, Naïve Bayes, Logistic Regression, Random Forest	KDD	PD
[40]		✓			K-means, Naïve Bayes	CAIDA	FD
[31]		✓			Random Forest	CAIDA	FD
[25]		✓			Bayesian Network	AMI	PD
[151]		✓			Optimized Cosine Similarity	Simulation	FD
[152]		✓			Rule-based attack detection	DARPA	PD
[153]		✓			Particle Swam Optimization	KDD	FD
[162]		✓			K-means	KDD	PD
[154]			✓		Rule-based attack detection	KDD	PD
[155]			✓	✓	Ensemble Clustering	MAWI	PD

3.9 Chapter Summary

In this chapter, we followed a systematic experimentation framework [156] to quantify the impact of four architectural tactics on the *accuracy* and *response time* of a BDCA system. Our findings quantify the impact of the strategies with respect to three contextual factors (i.e., security data, ML algorithm, and execution mode). Our findings reveal that contextual factors play an important role in the effectiveness of the evaluated tactics. Based on our findings, we formulate a set of design guidelines that can help researchers and practitioners in deciding when to use (and not use) the evaluated tactics in a BDCA system's design.

Chapter 4

A Tactics-driven Approach for Designing Big Data Cyber Security Analytics Systems

Related publication:

This chapter is based on our paper, which is under review in the *Journal of System and Software (JSS 2020)* "A Tactics-driven Approach for Designing Big Data Cyber Security Analytics Systems"[2].

In Chapter 2, we observed that architecting BDCA systems is a complex design activity, which involves critical decisions about the selection of architectural tactics for the satisfaction of various quality goals. Software architects need to consider associated dependencies, constraints, and impact on quality goals, which makes the design process quite challenging. Moreover, Chapter 3 shows that the effectiveness of architectural tactics depends upon several contextual factors, which further contributes to the complexity of BDCA design. To facilitate the design process, in this chapter, we first propose guidance models for supporting the systematic design of BDCA systems. We then present a heuristic-based approach (*TACTics*) that uses the guidance models to determine a system's architecture, which optimizes conflicting quality goals, dependencies, and constraints. We evaluate *TACTics* by designing and implementing candidate architectures for a Hadoop-based BDCA system. The findings of this chapter highlight the importance of several factors (e.g., selection of tactics, dependencies among tactics, and design constraints) that should be considered in designing a BDCA system.

4.1 Introduction

Chapter 2 reveals that (i) the state-of-the-art uses different architectural tactics (e.g., removal of duplicates and alert ranking) to achieve different quality attributes such as accuracy and response time in BDCA systems (ii) the tactics used in the design of BDCA systems have associated constraints, dependencies on other tactics, and impacts on multiple quality attributes and (iii) the selection and implementation of tactics varies among the reviewed BDCA systems. Afterward, Chapter 3 shows that the effectiveness of the tactics depends upon several factors such as the quality of security event data fed into a BDCA system, the employed algorithms, and the execution mode of a BDCA system (e.g., standalone or distributed). These findings underline the inherent complexities in the design of a BDCA system. Such complexities make it challenging to decide which set of tactics should be used to architect a BDCA system that can satisfy the quality goals, the associated constraints, and dependencies. We pose this problem as follows. "***How to determine an architecture for a BDCA system that best satisfy quality goals, dependencies, and constraints?***".

In this chapter, we present *TACTics*, a **T**actics-driven Design **A**pproach for Big Data **C**yber Security Analy**tics**, which builds on the work reported in Chapter 2 and optimization techniques to quantitatively determine an optimal architecture for a BDCA system that best satisfy quality goals, dependencies, and constraints. We first present guidance models for various BDCA design areas (e.g., data engineering). The guidance models map the problem space (requirements) to the solution space (tactics) and reveal the dependencies, constraints, quality implications, and implementation option for the tactics. We then present a heuristic-based design approach that uses the guidance models to generate, compare, and recommend an optimal architecture for a BDCA system. The design approach promotes a systematic, disciplined, and traceable BDCA design that takes into account important factors (such as quality implications and constraints) to determine an optimal architecture. We have evaluated *TACTics* by designing and implementing a Hadoop-based BDCA system. It is found that in 45% cases, the architecture found most optimal at runtime is the same as recommended at design time. Moreover, in 80% cases, the top five most optimal architectures at runtime includes the architecture recommended at design time.

Contributions: This chapter makes the following contributions

1. Presents guidance models to reveal quality implications, dependencies, and constraints for BDCA tactics (first part of *TACTics*)
2. Proposes a heuristic-based design approach for determining an optimal architecture for a BDCA system (second part of *TACTics*)
3. Demonstrates the effectiveness of the approach in a distributed setting through rigorous experimentation using four security datasets and five ML algorithms

Chapter Organization: Section 4.2 outlines the research approach followed for conducting the study reported in this chapter. Section 4.3 presents the guidance models followed by the design approach in Section 4.4. Section 4.5 presents the case study undertaken for the evaluation of *TACTics*. Section 4.6 outlines the threats to the validity of the findings reported in this chapter. Section 4.7 discusses the related work and finally, Section 4.8 concludes the chapter.

4.2 Motivation

This section motivates the need of tactics and guidance models for designing a BDCA system and emphasizes the importance of an optimal BDCA system's design.

4.2.1 Abstraction level and applicability of architectural tactics

Architectural tactics are reusable solutions built from experience to achieve a quality attribute. According to Bass et al. [33], "architectural tactics are means of satisfying a quality attribute response measure by manipulating some aspects of a quality attribute model through architectural design decisions". For example, Heartbeat is an architectural tactic for achieving reliability. Architectural tactics come in various shapes and sizes to achieve a wide range of quality attributes [33]. For instance, some tactics are defined at a higher abstraction level while others are defined at a more fine-grained level [1]. The tactics (e.g., Heartbeat, Ping/Echo, and Secure session) initially proposed in various studies (e.g., [33, 163, 164]) were applicable for all types of software systems. However, the increasing complexity of software systems motivated the researchers to propose tactics applicable for specific type of systems such as cyber-foraging [67] and cloud-based systems [68]. Furthermore, tactics-centric approaches are increasingly adopted in various domains to design software systems [165, 166]. A tactic

may be either optional or compulsory [138]. Optional tactics enable the architect to maneuver the overall system's architecture.

4.2.2 Need for BDCA guidance models

BDCA systems are complex software systems that are challenging to design due to relationships, dependencies, or interactions between their parts. Chapter 2 has identified and codified architectural tactics for achieving various quality attributes. However, the tactics do not work in a standalone fashion, rather interact with other tactics in the overall system' design. The relationship between tactics exposes the dependency between tactics and have associated constraints and impacts on quality attributes. For an architect, it is challenging to understand the relationship among the tactics, the impacts of tactics on quality attributes, and at the same time keep an eye on the associated constraints. Furthermore, the big data ecosystem offers a variety of tools for the implementation of the tactics. It is important that the implementation options for each tactic are known to the architects to better choose an option that suits their industry needs. Therefore, there is a need for guidance models that illustrate the relationship between tactics, the impacts of tactics on quality attributes, the constraints associated with each tactic, and the implementation options for the tactics.

4.2.3 Need for optimal BDCA design

We studied the architectures of BDCA systems presented in the 74 studies considered in our SLR (Chapter 2) to find out whether or not the selection of tactics varies among different architectures. We found that the selection of tactics significantly varies among the reviewed BDCA systems. For example, [158] and [155] leverage only three tactics (i.e., Parallel Processing, Signature-based Detection, and Anomaly-based Detection) in their BDCA system's design. On the other hand, [41]and [150] use four tactics i.e., Removal of Duplicates, Feature Selection, Parallel Processing, and Anomaly-based Detection to design a BDCA system. This finding stimulates a question: *What is is the impact of the selection of tactics on the performance of a BDCA system?* To illustrate the impact, we designed Hadoop-based BDCA architectures consisting of different set of tactics. The architectures are specified by the workflows (i.e., combination of tactics) presented in Table 4.5. Each workflow specifies a combination of tactics in a certain sequence. For example, workflow-3 in Table 4.5 indicates that the architecture of the BDCA system consists of three tactics i.e., (T2) Handling Missing Values, (T4) Parallel Processing,

Table 4. 1 Evaluation metrics and their descriptions

Metric	**Description**	**Equation**
Accuracy	Percentage of correctly detected attack and normal instances	$\text{Acc} = \left(\frac{TP+TN}{TP+FP+FN+TN}\right) \times 100$
F1 Score	The weighted harmonic mean of precision and recall	$\text{F1} = 2/\left(\frac{1}{Precision} + \frac{1}{Recall}\right)$
Detection Rate	Percentage of correctly detected attack instances	$DR = \left(\frac{TP}{TP+FP}\right) * 100$
False Positive Rate (FPR)	Percentage of normal instances detected as attack instances	$FPR = \left(\frac{FP}{FP+TN}\right) * 100$
Training Time	Time taken by the system in seconds to train the model using training data	
Prediction Time	Time taken by the system in seconds to detect attacks in the testing data	
True Positive (TP) = No. of correctly detected attack instances False Positive (FP) = No. of normal instances detected as attack instances False Negative (FN) = No. of attack instances detected as normal instances True Negative (TN) = No. of correctly detected normal instances $\text{Precision} = \left(\frac{TP}{TP+FP}\right) * 100$ $\text{Recall} = \left(\frac{TP}{TP+FN}\right) * 100$		

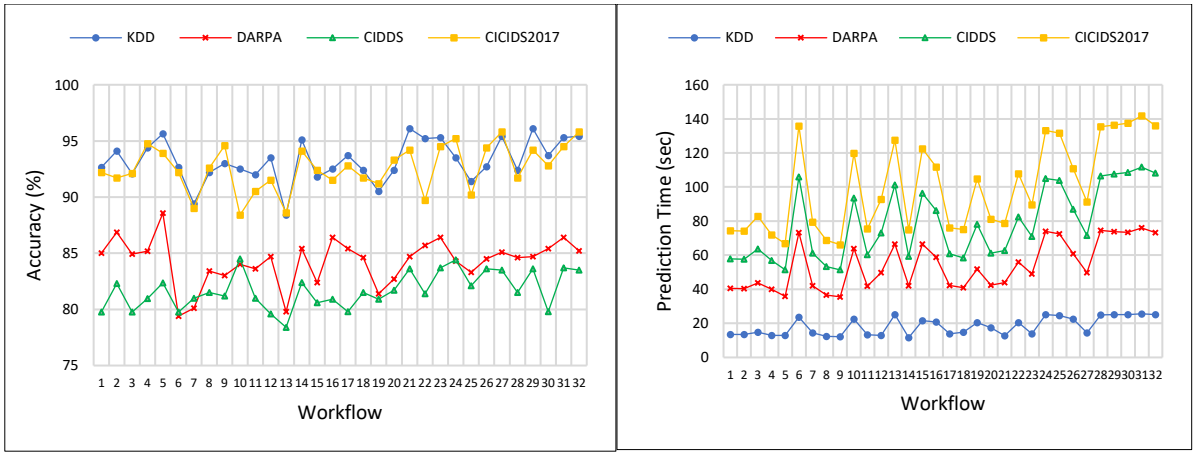


Figure 4. 1 Impact of workflow on the accuracy and prediction time of a BDCA system

and (T6) Anomaly-based Detection. After specifying the BDCA architectures via workflows, we implemented the workflows on a large-scale OpenStack cluster to investigate their impact on two most important quality attributes (i.e., accuracy and prediction time). The implemented architectures were evaluated using four security datasets: KDD [55], DARPA [56], CIDDS [57], and CICIDS2017 [58]. The metrics used for measuring accuracy and prediction time are presented in Table 4.1. The experimental details are available in Section 4.6. Figure 4.1 shows the accuracy and prediction time achieved for the 32 workflows and four datasets. It can be seen that the workflow achieving highest accuracy varies for the datasets. For instance, workflow 21 achieves highest accuracy (96%) for KDD while workflow 5 achieves highest accuracy (88%) for DARPA. Furthermore, a workflow achieving high accuracy for one dataset does not necessarily achieve high accuracy for another dataset. For example, workflow 21 achieving an accuracy of 96% for KDD merely achieves an accuracy of 84% for DARPA. The impact of the workflow on the prediction time also varies for the datasets. For example, workflow 14 achieves lowest prediction time for KDD but not for either of the other three datasets. It is also worth noting that a workflow achieving highest accuracy does not necessarily achieves the lowest prediction time. From these findings, we can conclude that the selection of tactics significantly impacts the performance of a BDCA system. Hence, the tactics should be selected in a way to generate an optimal BDCA design that can optimize the key quality attributes (e.g., accuracy, prediction time, and scalability etc.) of the system.

4.3 Research Approach

This section describes the approaches followed for developing the guidance metamodel, concrete guidance models, and the design approach.

4.3.1 Development of guidance metamodel

The metamodel characterizes the elements of the guidance model and the relationship among the elements. The metamodel has been developed based on the guidelines provided in [167] and taking inspirations from the state-of-the-art on guidance models in various domains such as Cyber-foraging [168] and Microservices [169]. Our metamodel is presented in Figure 4.2, which divides the design process into a problem space and a solution space. The problem space specifies the requirements (both functional and non-functional) and the solution space employs architectural tactics to address the requirements. The bold-headed arrow between a tactic and a requirement indicates that the tactic can be used to satisfy the requirement(s). Since BDCA systems use machine learning models to detect

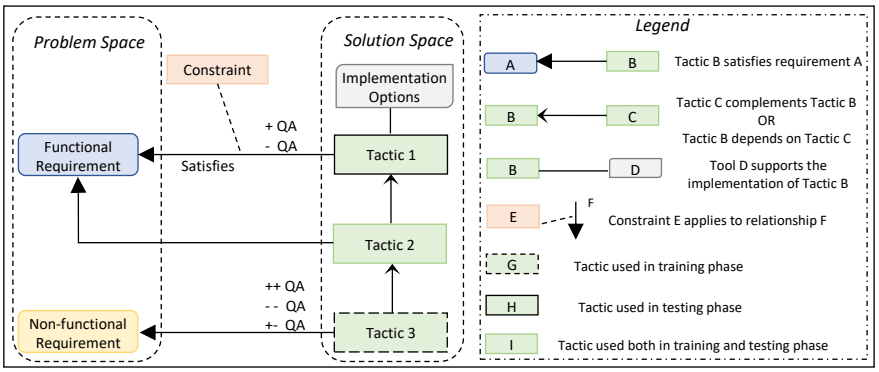


Figure 4. 2 Metamodel for understanding the concrete guidance models

attacks, the tactics are divided into three types based on whether the tactic is used in training phase, testing phase, or both. The tactics used in training phase are specified by green box with dotted border line, in testing phase by green box with solid border line, and in both training and testing phase by green box with no border line. If there are any risks or conditions associated to enable a tactic to satisfy the requirement, these are represented as constraints. A relation between a tactic and requirement(s) has some implications for various Quality Attributes (QA). These implications are denoted as + QA, - QA, ++ QA, -- QA, and +-QA in Figure 4.2. The '+' specifies a positive impact on a particular QA, '-' specifies a negative impact, '++' specifies a strong positive impact, '- -' QA specifies a strong negative impact, and '+-' specifies a potential for both positive and negative impact. Table 4.2 briefly describes the QA considered in the guidance models. The light-headed arrow between two tactics specifies that a tactic (tactic 1) requires or depends upon another tactic (tactic 2) for its incorporation in a design. In other words, a tactic (tactic 2) complements another tactic (tactic 1). By complementarity, we mean that a tactic supports another tactic in achieving its objectives. Although dependency and complementarity are two different sides of the same relationship, it is important to differentiate between dependency and complementarity. This is because dependency is an undesirable quality and complementarity is a desirable quality of a tactic. The line between a tactic and the associated implementation option specifies that the mentioned tool/technology can be used for the implementation of the tactic.

4.3.2 Development of concrete guidance models

We have developed concrete guidance models based on the knowledge gained through our systematic review [1] reported in Chapter 2 and experience gained in empirical study reported in Chapter 3.

Table 4. 2 Quality attributes considered in the guidance models and their descriptions

Quality Attribute (QA)	Brief Description
Accuracy	The degree to which a BDCA system provides correct answer with the required degree of precision
Training Time	The time taken by a BDCA system (in seconds) to train a machine learning model using the training data
Prediction Time	The time taken by a BDCA system (in seconds) to detect attacks in the testing data
Scalability	How easily a BDCA system can grow to handle increasing workload
Reliability	How long the a BDCA runs without experiencing a failure
Security	How well a BDCA system protects itself and its data from unauthorized access
Usability	How easy it is for people to learn, remember, and use a BDCA system
Reusability	How easy it is to use components of a BDCA system in other systems
Generality	The range and types of cyber attacks detectable through a BDCA system
Interoperability	How easily a BDCA system connects and exchanges data with other systems
Complexity	How easy it is to understand/verify the design and implementation of a system and its parts
Cost	How much it costs in terms of money to develop and/or operate a BDCA system

4.3.3 How to use the guidance models

At an abstract level, the functional requirements of a BDCA system are divided into five phases (i.e., data engineering, feature engineering, process engineering, data processing, and data post-processing) in the form of a big data lifecycle as shown in Figure 4.3. First, tactics are selected to address these functional requirements/phases, which are then accompanied by tactics for satisfying the associated non-functional requirements (e.g., scalability and reliability).

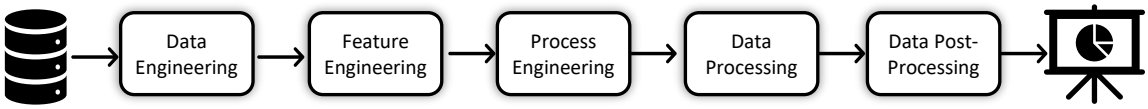


Figure 4. 3 BDCA design phases

4.3.4 Heuristic-based design approach

The guidance models provide guidance for designing a BDCA system. However, as revealed in Section 4.2.3, the BDCA design also needs to be optimal. Therefore, we also propose a heuristic-based approach to determine an optimal design for a BDCA system. The approach is based on the design knowledge obtained from the guidance models. The design knowledge is underlined in the form of five design metrics, which are used in the heuristic-based approach to determine an optimal BDCA design. The design metrics are: (i) *tactic's dependencies* (ii) *tactic's constraints* (iii) *tactic's positive impacts on QAs* (iv) *tactic's negative impact on QAs* and (v) *tactic's complementarity*. The design metrics are already elaborated in guidance metamodel (Section 4.3.1).

4.4 Guidance Models

This section presents concrete guidance models for BDCA design phases shown in Figure 4.3.

4.4.1 Data engineering

Figure 4.4 presents the guidance model for data engineering. The tactics instantiated during the data engineering aim to pre-process and clean the collected data. The **Removal of Duplicates** tactic removes duplicate records from the training data to ensure that Machine Learning (ML) model is not biased towards learning more frequent behaviours. The existence of a large number of duplicates prevents a model from learning the rare behaviours that indicate dangerous attacks [170]. The tactic *reduces training time* by reducing data size and *improves accuracy* by ensuring the development of an unbiased ML model. The tactic requires that the functionality responsible for removal of duplicates should not be computationally expensive. The tactic can be implemented using the *NullWritable* class in Hadoop library. The **Handling Missing Values** tactic detects the records with the values missing for some features (e.g., protocol type). Such records create a substantial amount of bias for subsequent data analysis. Upon detection, the tactic either removes the record or implants a new value for the feature using the data imputation technique. The tactic mainly *improves accuracy* but *increases the training time*. Some common tools for data imputation include NORM, SOLAS, and SAS. The **Removal of Incorrect Data** tactic discards records with future values that are not suitable for security analytics. For example, '-20' is not a suitable value for number_of_bytes feature in a record. The tactic *improves accuracy* with a negligible *increase in training time*. The tactic requires careful formulation of rules (e.g.,

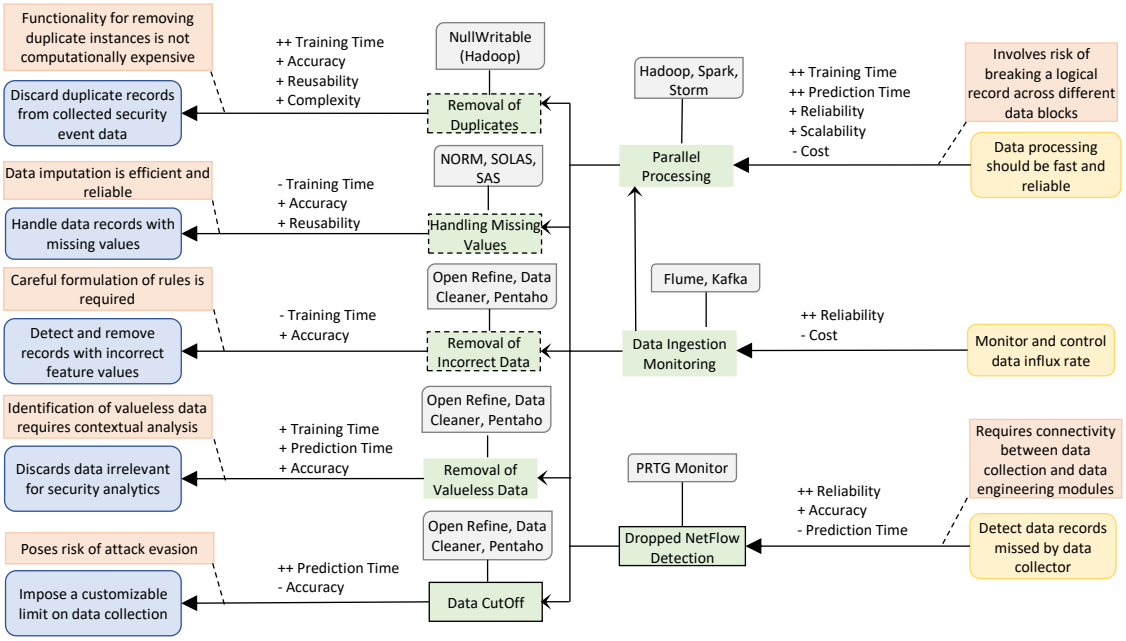


Figure 4. 4 Guidance model for data engineering phase of a BDCA system

number_of_bytes > 0) for detecting incorrect values. The **Removal of Valueless Data tactic** discards data (from the collected pool of data) that does not contribute to detecting attacks. For instance, zero-byte flows are collected by a network sniffer; however, such data is only used in TCP/IP handshaking and does not have any relevance to attack detection [171]. Similarly, image data captured as URL suffixes with JPEG or JPG needs to be discarded [172]. The tactic *reduces training time* by reducing data size. The **Data CutOff tactic** imposes a customizable limit on data to be collected for each network connection or a process. For example, only collecting first 15 Kbytes of a network connection or storing data pertaining only to the first 100 sec of the execution of a process. This tactic considerably reduces the size of data in order to *reduce training time*. However, the tactic carries a risk of evading an attack if an attacker plans the execution of an attack after the customizable limit [42]. The separation of code as an individual tactic *improves a system's reusability and modularity*. All five tactics associated with functional requirements depends on Parallel Processing tactic to manage the distribution of processing among computing nodes and Data Ingestion Monitoring tactic to prevent node failures by controlling data influx into a system. The **Parallel Processing** tactic *improves response time, scalability, and reliability*, however, *increases the operation cost* of a system. The tactic involves a minor risk of breaking a record across different blocks during data partitioning. The tactic depends on Data Ingestion Monitoring tactic and Dynamic Load Balancing tactic for preventing node crash and balancing the load among nodes. The tactic is implemented using big data frameworks such as Hadoop [50], and Spark [51]. The **Data Ingestion Monitoring** tactic substantially *improves reliability* by keeping a constant check on data ingestion rate, however, complicates the deployment of a system. The tactic is implemented using Flume or Kafka servers. **The Dropped NetFlow Detection** tactic detects records that are missed by a data collection tool (e.g., Wireshark). The tactic monitors the sequence number of records and if found out of order, a message is flagged. The tactic *improves reliability* by ensuring data is collected in its entirety and *increases the prediction time* due to the monitoring process. The tactic requires connectivity between data collection and data engineering modules. PRTG monitor [173] is a well-known tool for implementing the tactic.

4.4.2 Feature engineering

Figure 4.5 presents a guidance model for feature engineering. The tactics used in feature engineering aims to select, generate, and/or transform features in the collected security event data. The **Feature Selection** tactic selects the most relevant features out of the several available features in data. The security event data often consists of many features as evident from 41 features in KDD [55] and 82 features in CICIDS2017 [58] dataset. Not all of the features contribute to attack detection. Therefore, the irrelevant features are discarded to reduce data size and complexity, which eventually leads to *improved accuracy* and *reduced training and prediction time*. However, discarding features in situations where the number and nature of features continuously change (e.g., in dynamic networks) can harm the accuracy of a system. The **Feature Generation** tactic generates new features from the existing features to either improve accuracy or reduce data dimensionality. For example, Rathore et al., [41] sum source_bytes and destination_bytes to generate a new feature number_of_bytes. The tactic *improves accuracy*, however, the impact on training and prediction time depends upon whether the overall number of features is increased or decreased after the feature generation. An increase in the number of features increases training and prediction time and vice versa. This tactic *increases the overall complexity* of a system and requires a light-weight feature generation mechanism to achieve improvement in training and prediction time. The **Feature Transformation** tactic enhances security data in a way that an ML model can efficiently analyse data to detect suspicious cyber activities. For instance, the features' values can be transformed into a range of 0 to 1 to help an ML model to discriminate between malicious and benign access request. This tactic *improves accuracy but increases training and prediction time* and *complexity*. Some common techniques for implementing this tactic includes normalization, bucketing, and one hot encoding. All of the three tactics associated with the functional requirements depends on Parallel Processing and Maintaining Multiple Copies tactics. The Parallel Processing tactic distributes the processing for feature engineering among nodes. The **Maintaining Multiple Copies** tactic stores and maintains multiple copies (three by default) of the data processed in the feature engineering phase. These choices address the *issue of node failure* and *improve training and prediction time*. This tactic increases the attack surface by exposing the data on multiple fronts and *increases storage cost*. The tactic requires connectivity among nodes hosting data replicas and may introduce data inconsistencies. The tactic relies on **Heart Beat** tactic to check connectivity and health of the nodes sharing data replicas.

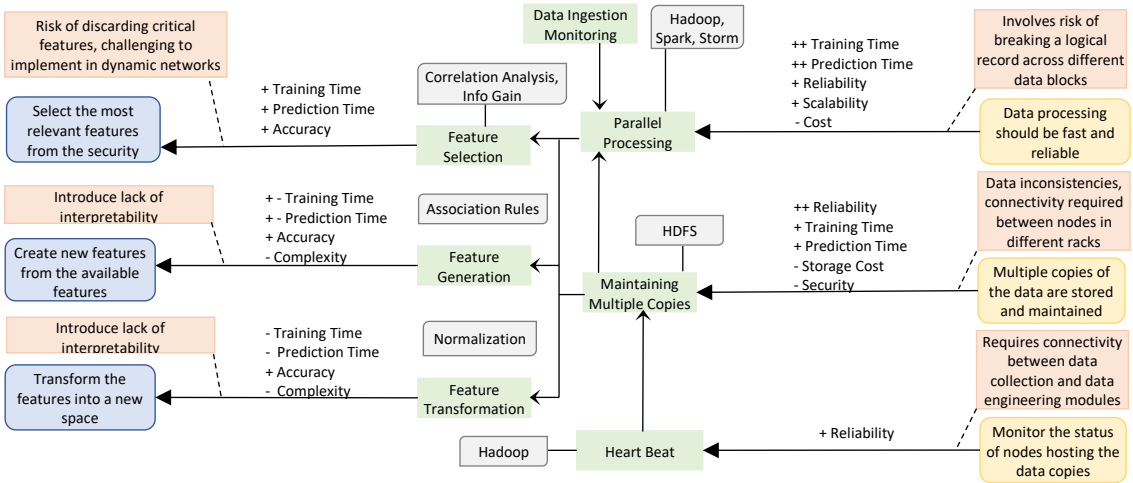


Figure 4. 5 Guidance model for feature engineering phase of a BDCA system

4.4.3 Process engineering

The process engineering phase ensures effective data processing in the subsequent phase (i.e., data processing). The guidance model for process engineering is shown in Figure 4.6, which employs two tactics (i.e., ML Algorithm Selection and Parameter Tuning). The two most important factors for effective security analytics are input data quality and the employed ML algorithm [59]. Therefore, **ML Algorithm Selection** tactic assesses the applicability of various ML algorithms (e.g., Naïve Bayes and K-means) to select the most suitable algorithm to be employed in the subsequent phase. The algorithms are assessed based on their time complexity, incremental update capacity, attack detection accuracy, online/offline mode, and generalization capability. The tactic *improves accuracy and usability* by ensuring to select the algorithm that generates highly accurate and interpretable results. The impact on training and prediction time depends upon the time consumed in the selection process vs the time gained from the efficiency of the selected algorithm. Moreover, the tactic *increases the overall complexity* of the BDCA system. The tactic requires careful trade-off among various quality attributes [59]. For example, an algorithm may generate more accurate results but at the cost of increased prediction time. The tactic depends upon Removal of Duplicates, Handling Missing Values, and Feature Selection to implement the minimum preprocessing required for the collected data. The Parallel Processing tactic supports the tactic to distribute the processing among the nodes. Distributed ML libraries (such as Apache Mahout [52] and MLlib [51]) are used to implement and assess various algorithms. The **Parameter Tuning** tactic tunes the parameters (e.g., weights in a regression model) and hyperparameters (number of clusters in K-means) for the algorithms and frameworks (e.g., Hadoop). In addition to ML algorithms, parameter tuning is very important for the underlying frameworks. For example, the number of Map and Reduce jobs, buffer size, and the number of data replicas need to be configured for Hadoop. This tactic *improves accuracy* with potential for both increase and decrease in training and prediction time. Furthermore, it *decreases the usability* of a system as a user has to manually or semi-manually tune the parameters. The tuning of parameters requires user expertise. Some techniques for ML parameter tuning include Grid search and Bayesian optimization.

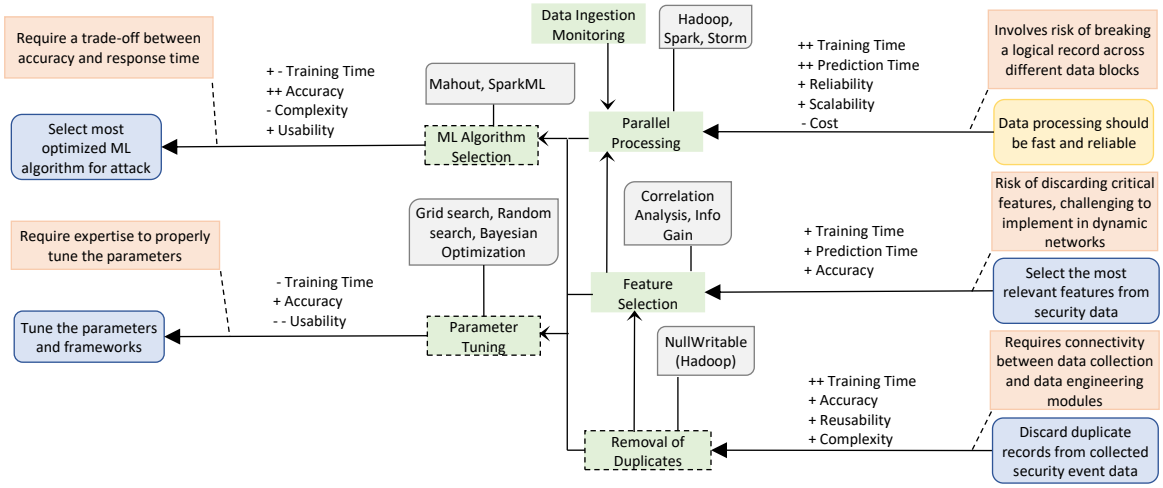


Figure 4. 6 Guidance model for process engineering phase of a BDCA system

4.4.4 Data processing

The guidance model for data processing is shown in Figure 4.7. This phase uses two tactics for detection of attacks. The **Anomaly-based Detection** tactic is the most crucial tactic, which uses an ML model to

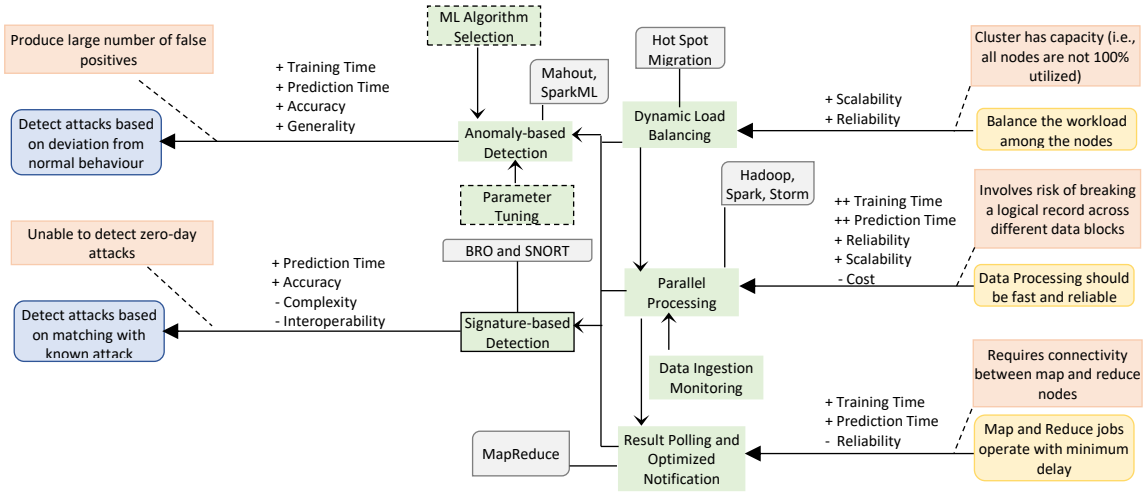


Figure 4. 7 Guidance model for data processing phase of a BDCA system

detect cyber behaviour that deviates from normal behaviour. Anomaly-based detection is very effective in detecting zero-day attacks [174]. The efficient use of this tactic *improves accuracy, training and prediction time, and generality* of a BDCA system. The tactic relies on ML algorithm selection and Parameter Tuning tactics to ensure effective selection and execution of ML algorithms and frameworks. The key disadvantage of this tactic is the potential to *generate a large number of false positives*. Apache Mahout [52], MLlib [51], and Distributed Weka are some of the libraries used for implementation of ML algorithms used in anomaly detection. The **Signature-based Detection** tactic detects known attacks based on their attack signatures. The tactic *improves accuracy* only for known attacks by reducing false alarms and is faster as compared to anomaly-based detection. However, this tactic not only *increases the overall complexity* but also poses a *challenge of interoperability* between anomaly-based detection and signature-based detection. This concern is more severe in cases where a signature-based detection is an open-source tool such as Zeek (BRO IDS)[12] and SNORT[13]. Signature-based detection is unable to detect zero-day attacks. Moreover, this tactic requires frequent updates of attack signatures to be able to detect newly introduced attacks. For updating the signatures, this tactic uses **Attack Pattern Updation** tactic that (semi) automatically updates the signatures used by the signature-based detection tactic. In addition to Parallel Processing and Dynamic Load Balancing, both the tactics in data processing phase depends upon **Result Polling and Optimized Notification** tactic, which optimizes the delay in feeding results from mapper nodes to reducer nodes. This way Mapper nodes do not have to wait for predefined time limit rather they can notify the Reducer as soon as the Mapper results are ready. This tactic *improves prediction time* but makes nodes more vulnerable to failure. The tactic is incorporated in a Parallel Processing setup that uses MapReduce framework.

4.4.5 Data post-processing

The guidance model for data post-processing is presented in Figure 4.8. This phase processes the generated alerts to remove (any) false positives, correlate the alerts to reveal sophisticated attacks, and rank them to facilitate users in responding to them. The tactics pertaining to functional requirements are as follows. The **False Positive Reduction** tactic uses mining techniques (e.g., clustering and

[12] https://www.zeek.org

[13] https://www.snort.org

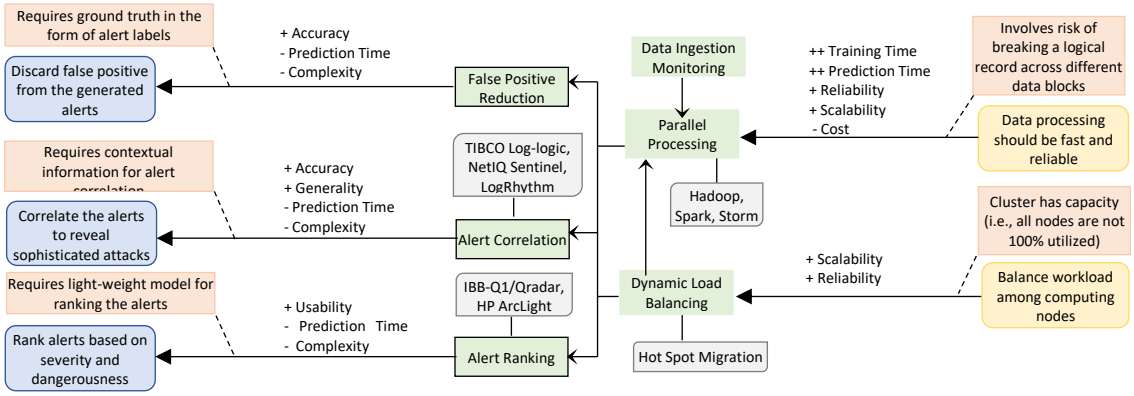


Figure 4. 8 Guidance model for data post-processing phase of a BDCA system

classification) to learn about true positive and false positive from various features (e.g., IP address and protocol) of alerts [175]. The learnt characteristics are then used to classify future alerts. This tactic *improves accuracy* but *increases prediction time* due to extra processing. The tactic requires ground truth in the form of labels to mine insight from previous alerts. The tactic depends upon Anomaly-based Detection tactic for alert generation. The **Alert Correlation** tactic correlates alerts based on logical relations to generate attack scenarios that help reveal sophisticated attacks (e.g., Advanced Persistent Threats) and root cause of attacks [175]. This tactic *improves accuracy and generality* at the cost of *prediction time* and complexity of a BDCA system. This tactic requires contextual information (e.g., network topology) to correlate alerts. Moreover, missing links and/or pre and post alerts can be a hurdle in efficient alert correlation. The tactic uses alerts generated by Anomaly-based Detection and Signature-based Detection tactics. Some commercial tools for alert correlation include TIBCO Log-logic, NetIQ Sentinel, and LogRhythm [175]. The **Alert Ranking** tactic ranks the final list of alerts based on their dangerous and severity. The ranking facilitates users to respond to severe alerts on a priority basis, which *improves a system's usability*. This tactic *increases prediction time* due to the processing involved in ranking alerts. The tactic should use a lightweight ranking model otherwise the intensive ranking processing cancels the benefit gained through prioritized response to severe alerts. Some example tools for ranking alerts are IBB-Q1/QRadar and HP ArcSight [175]. All the three tactics in post-processing require Parallel Processing tactic to speed up the processing and Dynamic Load Balancing tactic to balance the load among nodes and prevent node failures.

4.5 Heuristic-based Design Approach for BDCA

In this section, we present the goal of the design approach, the design metrics, and the detailed steps of the design approach for determining an optimal BDCA architecture.

4.5.1 Goal of the approach

The guidance models reported in the previous section of the chapter present several tactics for designing a BDCA system. However, all these tactics do not necessarily need to be incorporated in designing a BDCA system. This is because the effectiveness of a BDCA system depends upon several contextual factors such as quality and quantity of security data, ML model employed, and execution mode (fully distributed or pseudo-distributed) of a system as illustrated in Chapter 3. For example, incorporating the Removal of Duplicates tactic in an environment that does not generate duplicate records is counterproductive. Similarly, the impacts of the tactics on quality attributes varies with these contextual factors. For instance, the use of Feature Selection tactic has a negative impact on response

time and a negligible positive impact on accuracy when the data has a very few number of features [176]. Furthermore, the importance of quality attribute achievement also varies. For example, response time is less of a concern in a setup with a large computational resource. All these factors motivate the need for a design approach that embodies a software architect's experience to determine an optimal design for a BDCA system. Therefore, our design approach (*TACTics*) *aims to exploit the guidance models to produce/determine a (near) optimal design for a BDCA system*. By near optimal, we mean a design that maximizes positive design metrics (e.g., positive impact on quality attributes and tactics' complementarity) and minimizes negative design metrics (e.g., negative impacts on quality attributes and tactics' dependencies).

4.5.2 Design metrics considered in *TACTics*

In order to select the (near) optimal architecture from the set of potential architectures, *TACTics* uses five design metrics revealed through the guidance models discussed in Section 4.3.1. These metrics include: (i) *tactic's dependencies* (ii) *tactic's constraints* (iii) *tactic's positive impacts on QA* (iv) *tactic's negative impact on QA* and (v) *tactic's complementarity*. These metrics evaluate the pros and cons of each tactic to quantify the overall optimality of an architecture. The values for the metrics are determined from the guidance models.

4.5.3 Design approach overview

TACTics consists of four steps as shown in Figure 4.9. *Step 1:* starts with the first tactic and determines the values for the five metrics associated with the tactic from the guidance models. *Step 2:* assigns weights to each metric (e.g., positive QA) to specify the importance of the metric and calculate the

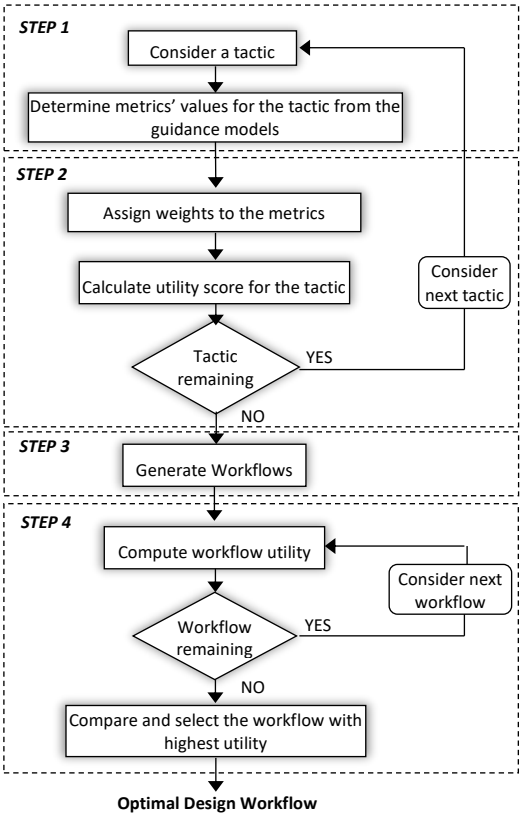


Figure 4. 9 BDCA design approach (*TACTics*) overview

overall utility of the tactic. *Step 3:* generate the workflows and *Step 4:* calculates the utility of the workflow consisting of the different set of tactics to determine the most optimal workflow for the system's design. The utility score of the workflow is calculated by combining the utility scores of the individual tactics that are part of the workflow. It is important to note that a tactic having high utility score does not necessarily mean to be part of the most optimal workflow. This is because our approach selects a workflow based on the overall utility score instead of individual tactics' utility scores. Furthermore, it is also possible that a tactic has a high utility score but if it depends on another tactic that is not present in the workflow, the workflow is discarded and hence the tactic (with high utility score) is not included (Algorithm 4.1).

4.5.3.1 Step 1: Metric score determination

Metric score is the value of a metric (e.g., dependencies) for a tactic obtained from the guidance model. For example, the value for dependency metric for Feature Selection tactic is 2 as the tactic directly depends upon two tactics (i.e., Parallel Processing and Maintaining Multiple Copies) as shown in Figure 4.5. This step determines the values for the five metrics for each tactic. This is important because these scores are indicators of the utility of the tactic. For example, a tactic having a high score for dependency and a low score for complementarity is less effective as compared to a tactic with low dependency score and high complementarity score.

4.5.3.2 Step 2: Weighting metrics

Step 1 assigns scores to various metrics, however, the significance of one metric might be greater than another. For example, the positive impact of a tactic on a QA might be more severe than its negative impact on another QA. Therefore, step 2 requires a software architect to assign weights to each metric. To facilitate the assignment of weights, in the following, we provide guidelines with respect to each tactic and associated metrics. These guidelines are formulated based on (i) our theoretical understanding of the domain of BDCA by conducting a large-scale systematic literature review [1] of BDCA systems reported in Chapter 2 and (ii) our observations from a large-scale empirical study [6] on a BDCA systems reported in Chapter 3, which quantifies various impacts of the considered tactics.

Guidelines for assigning weights: In the following, we provide general guidelines for the assignment of weights to the metrics associated with each tactic.

Data Engineering Phase: We propose the following guidelines for assigning weights to the tactics in data engineering phase. ***Removal of Duplicates:*** The effectiveness of this tactic primarily depends upon the quality of the security events data. If the data contains a large number of duplicate instances, the positive impact on QAs (e.g., accuracy and response time) and complementation ability of this tactic is quite high. Hence, high weight should be assigned. The dependencies on other tactics (e.g., parallel processing) increases with the increase in the size of the data. ***Handling Missing Values:*** If the values of critical features (e.g., IP address) are missing, the importance of this tactic for its impact on QAs is quite high, hence, a high weight should be assigned. The number of instances having missing values also impacts the effectiveness of this tactic with more the number of instances with missing values, the higher should be the weight assigned for its impacts on QAs. The importance of complementing other tactics (e.g., ML algorithm selection) increases when the increase in the number of instances with missing values. ***Removal of Incorrect Data:*** If the values are incorrect for the critical features and a considerably large number of instances, the importance of this tactic for its impact on QAs increases. If

the incorrect values are not evident, i.e., it is not clear which value is incorrect, a high weight should be assigned for constraints such as careful formulation of rules for the identification of incorrect values. ***Removal of Valueless Data:*** The identification and removal of valueless data from the collected data (e.g., network traffic) are computationally expensive, therefore, this tactic heavily depends upon the parallel processing tactic and data ingestion monitoring for controlling the inflow of the data. The impact of this tactic on QAs is high if the BDCA is receiving data from a source that generates a large proportion of valueless data. ***Data CutOff:*** The importance of this tactic for its impact on QAs is high when accuracy is a greater concern as the tactic poses a clear risk of evading attacks. The tactic's dependency on parallel processing becomes critically important in situations where sufficient computational power is not available for data processing. ***Parallel Processing:*** This tactic is especially important for its impact on QAs such as response time and reliability when the data is of a large size that cannot be processed using a single machine. The implementation of this tactic requires expertise in configuring and running big data frameworks such as Apache Hadoop and Apache Spark. In case of a lack of such expertise, a high weight should be assigned to the associated constraints. ***Data Ingestion Monitoring:*** The importance of the tactic for its impact on QA such as reliability increases with the increases in the influx rate of the data. If the influx rate is high, a higher weight should be assigned to the tactic for its impact on QA. The tactic requires a server such as Flume to control data influx. In case of lack of expertise and/or cost concerns, a higher weight is to be assigned to the constraints associated with the tactic. ***Dropped NetFlow Detection:*** If the data influx rate is high, there is a probability of NetFlow getting dropped and hence attacks go undetected. Therefore, a high weight should be assigned to the tactic for its impact on QAs such as reliability and accuracy.

Feature Engineering Phase: We propose the following guidelines for assigning weights to the tactics in feature engineering phase***. Feature Selection:*** If the data contains a large number of features (e.g., more than 10 features), this tactic becomes important for improving accuracy and response time. If architects do not have the domain knowledge (e.g., which features are more or less important) in terms of attack detection, the importance of constraints associated with this tactic is high. In case of employing a computationally heavy algorithm for feature selection, a high weight should be assigned to the dependencies on other tactics such as parallel processing. ***Feature Generation:*** The importance of this tactic is high when the employed ML model is not able to detect attacks with higher accuracy. In such a case, this tactic should be used to generate extra features for model training. If the data has a small number of features (e.g., less than 10 features) and the majority of the features are numerical, the importance of this tactic for its impact on accuracy is increased. When the interpretability of the results generated by an ML model is a concern, a high weight should be assigned to the constraints associated with this tactic. ***Feature Transformation:*** If the variation between the values of various features in data is high, the importance of this tactic for its impact on accuracy increases. Chances of such a variation are higher in cases where the majority of the features are numerical. Therefore, the importance of this tactic increases for data with a large number of numerical features. ***Maintaining Multiple Copies:*** In case, data storage capability is available, a lower weight should be assigned to this tactic for its associated constraints. When the data is distributed and strong connectivity can be maintained between the nodes hosting the data, the importance of this tactic for its impact on QAs (e.g., reliability) increases. ***Heart Beat:*** When the replication factor of the data is more than one, the importance of this tactic for its impact on QAs increases. The associated constraints (e.g., the connection between data collection and data engineering modules) are quite important and should be given high weight in cases where the modules are hosted by different computing clusters.

Process Engineering Phase: We propose the following guidelines for assigning weights to the tactics in process engineering phase. ***ML Algorithm Selection:*** In cases where enough computational power is available, low weight should be assigned for dependencies and high weight for the impact of this tactic on QAs such as accuracy. This tactic also relates to the size of the training data where the impact on the response time increases with the increase in the size of the training data. ***Parameter Tuning:*** This tactic heavily depends upon parallel processing tactic as repetitive training for finding the optimal parameter configuration is computationally quite expensive. In cases where ML expertise is not available, a high weight should be given to the constraints associated with this tactic.

Data Processing Phase: We propose the following guidelines for assigning weights to the tactics in data processing phase. ***Signature-based Detection:*** This tactic complements the anomaly-based detection tactic to increase the accuracy of a BDCA system. For data with a large number of features, the anomaly-based detection tactic is already effective, therefore, the importance of signature-based detection for improving accuracy is reduced. This tactic heavily depends upon parallel processing tactic as the incorporation of this tactic requires two-phase processing. For the large size of training data, low weight should be given for the impact of the tactic on QAs such as accuracy and prediction time. ***Result Polling:*** This tactic is important in distributed setups where the map and reduce jobs are hosted by different machines. In such a case, the significance of this tactic for its impact on QAs, such as response time, increases. The implementation of this tactic requires an in-depth understanding of the MapReduce programming model to make the required changes. In case of a lack of such expertise, high weight should be given to the associated constraints. ***Dynamic Load Balancing:*** High weight should be assigned to the impact of this tactic on QAs such as reliability in cases where the storage and processing of the data are fully distributed so that the tactic can balance the load among the nodes. In cases where the computing nodes are vulnerable to getting into poor health condition, the importance of this tactic increases. The importance of this tactic for its impact on reliability also increases with the increase in the load on the computing cluster.

Data Post-processing Phase: We propose the following guidelines for assigning weights to the tactics in data post-processing phase. ***False Positive Reduction:*** When attending to false alarms is a greater concern, high weight should be given to the impact of this tactic on QAs. The tactic also requires an extra phase of data processing, which requires computational power. Therefore, the importance of parallel processing is increased especially in cases where the size of test data is quite large. ***Alert Correlation:*** Alert correlation is effective when the data has a large number of features and there exists some type of correlation among the features. Hence, this tactic becomes important for its impact on QAs such as accuracy when the data has a large number of features. This tactic depends heavily on other tactics like parallel processing and dynamic load balancing when the size of the test data is large. ***Alert Ranking:*** When there exists a large variance among the values of the features used for ranking (e.g., source bytes and destination bytes), the importance of this tactic for its impact on QAs increases. This is because large variation implies that alerts are of diverse nature and it is important to prioritize response to more dangerous alerts.

Using the above the guidelines, weights are assigned to the five metrics for each of the tactic. We then use Equation 4.1 to calculate the overall weighted score of the tactic. In Equation 4.1, 'T' denotes the tactic, $z \in [1, 21]$ specifies the tactic number, $M \in [0, 1]$ specifies the value of the metric, and 'm' specifies the total number of metrics.

$$T_z = \sum_{j=1}^{m} M_j \tag{4.1}$$

$$WF_N = \sum_{g=1}^{a} \binom{a}{g} + \sum_{g=1}^{a} \binom{a}{g} \sum_{h=1}^{b} \binom{b}{h} \tag{4.2}$$

$$W_x = \sum_{z=1}^{k} T_z \tag{4.3}$$

4.5.3.3 Step 3: Workflow generation

Our aimed design of a BDCA system is driven by workflow that integrates a set of tactics for the composition of a system's design. The workflows are generated based on various combinations of tactics. The guidance models presented in Section 4.4 consists of a total of 22 tactics (Figure 4.10). Among these tactics, Anomaly-based Detection is considered as the core of a BDCA system [1]. Therefore, every workflow is expected to have Anomaly-based Detection tactic. The rest of the tactics can be operationalized or dropped as mentioned in Section 4.5.1. In order to generate the total number of possible workflows, we first order the tactics in terms of their position in a system with reference to Anomaly-based detection tactic as shown in Figure 4.10. We then use Equation 4.2 that uses mathematical combinations to calculate the total number of possible workflows, where WF_N denotes the total number of workflows, $a \in [0, 18]$ specifies the number of tactics before and $b \in [0, 3]$specifies the number of tactics after the Anomaly-based Detection tactic, g and h specifies the combination of elements from the total set. Given the number of tactics, the total number of possible workflows is quite large, which makes it challenging to analyse such large number of workflows. Therefore, we designed Algorithm 4.1 to reduce the number of workflows based on dependency analysis among tactics. According to Algorithm 4.1, a workflow that contains a tactic, where its complementary tactic, is not available is discarded. For example, Feature Selection tactic depends upon Parallel Processing tactic. If a workflow contains Feature Selection tactic but not Parallel processing tactic, the workflow is discarded.

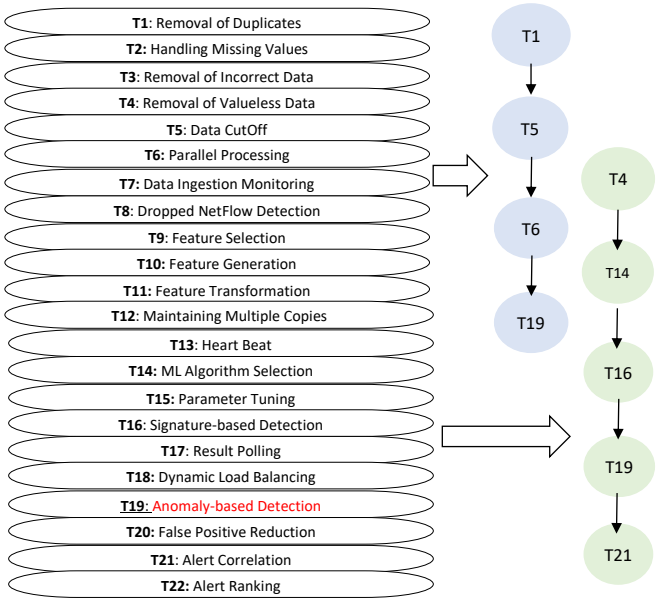


Figure 4. 10 Positioning of tactics in a BDCA system's design and sample workflows

Algorithm. 4.1. Workflows reduction algorithm

Input: $WF = \{WF_1, WF_2, \ldots, WF_{N}\}$ // *Set of workflows*
$T = \{T_1, T_2, \ldots, T_M\}$ // *Total number of tactics*
$D = \{(D_{11}, D_{12}, \ldots, D_{1K}), (D_{21}, D_{22}, \ldots, D_{2K}), \ldots, (D_{M1}, D_{M2}, \ldots, D_{MK})\}$ // *Dependencies for each tactic*
Output: WF_R = Reduced set of workflows
Steps:

```
1.  For i = 0 to i = N
2.      For j = 0 to j = WFi.. length
3.          if Tj = Anomaly-based Detection
4.              Jump to next tactic
5.          end
6.          else
7.              For z =0 to z = Dj.length
8.                  if Dj [z] ∉ WFi
9.                      Discard WFi
10.                 end
11.             end
12.     end
13. end
```

4.5.3.4 Step 4: Workflow selection

We calculate the utility of each workflow comprised of various tactics. The objective of this step is to select the most optimal workflow (as per our goal) for the composition of system's design. We use Equation 4.3 for calculating the utility score for each workflow, which sums up the utility scores for individual tactics involved in the workflow. In Equation 4.3, 'W' specifies the workflow, $T \in [0, 5]$ specifies the utility score of a tactic, and $k \in [1, 21]$ denotes the number of tactics in the workflow. After calculating the scores, the workflow with the highest utility score is selected for system's composition. This is the workflow that is composed of tactics that maximize the impact of positive metrics (e.g., complementarity) and minimize the impact of negative metrics (e.g., dependencies).

4.6 Case Study

In this section, we present the design of our case study, the application of *TACTics* in the case study, and the evaluation results of the case study.

4.6.1 Case study design

BDCA system: We apply *TACTics* to design a BDCA system for detecting various types of cyber intrusions such as DoS and Brute Force. The system monitors incoming network traffic and flags an alert upon detection of a malicious intrusion [59]. ***Tactics:*** Our study considers seven tactics for designing the system. These tactics are (T1) Removal of Duplicates (T2) Handling Missing Values (T3) Feature Selection (T4) Signature-based Detection (T5) Parallel Processing (T6) Anomaly-based Detection and (T7) Alert Ranking. ***ML Algorithms:*** We formulated anomaly detection as a binary classification problem with one normal class (benign network traffic) and one anomaly class (attack instances). As mentioned in Section 4.4.4 that Anomaly-based Detection uses an ML algorithm to detect deviations from normal cyber behaviour, we separately use five state-of-the-art algorithms. These algorithms include K-Means (KM), XGBoost (XGB), Naïve Bayes (NB), Support Vector Machine (SVM), and Random Forest (RF). ***Datasets:*** We use four well-known security datasets that are fed into the system to detect attacks in the dataset. The datasets are: KDD [55], DARPA [56], CIDDS [57], and CICIDS2017 [58]. The details of the datasets are available in Chapter 3.

4.6.2 BDCA system's design process

In this section, we applied the steps of *TACTics* to determine an optimal system's design.

4.6.2.1 Step 1: Metric score determination

We refer to our guidance models to determine the values of five design metrics for the seven tactics. The values for the tactics (shown in Table 4.3) are directly determined from the guidance models as exemplified in Section 4.5.3.1.

Table 4. 3 Values of design metrics determined from the guidance models

Tactic #	Tactic	Dependency	Complementarity	Constraints	Positive QA	Negative QA
T1	Removal of Duplicates	3	2	1	3	1
T2	Handling Missing Values	3	0	1	2	1
T3	Feature Selection	2	2	2	2	1
T4	Parallel Processing	3	16	2	3	1
T5	Signature-based Detection	3	0	1	2	2
T6	Anomaly-based Detection	5	0	1	3	0
T7	Alert Ranking	2	0	1	1	2

4.6.2.2 Step 2: Weighting metrics

The weights assigned to each of the five metrics for each tactic are presented in Table 4.4. The weights are assigned to the design metrics based on the guidelines for assignment of weights. For example, the positive impact of Removal of Duplicates on QA is high for KDD as compared to its impact for CICIDS2017. This is because KDD contains around 78% duplicate instances as compared 0.2% duplicates in CICIDS2017. The dependency score of Removal of Duplicates is quite strong for supporting the large-scale processing; therefore, a high weight is assigned. As another example, the impact of Feature Selection tactic on QA is weighted high for CICIDS2017 as compared to its low weight for CIDDS. This is because each record in CICIDS2017 has 82 features (as compared to 12 features in CIDDS), which brings the response time down by a significant margin. Our previous study suggested a significant impact of Signature-based Detection on response time for datasets with a high proportion of attack instances. Hence, a high weight is assigned to KDD (which contains 80.5% attack instances) as compared to a low weight for CICIDS2017 (which contains only 15.4% attack instances). It is important to note that weights are not assigned for Anomaly-based Detection tactic. This is because Anomaly-based detection tactic should be part of every workflow as mentioned in Section 4.5.3.3. After assigning weights, we calculate the score for each metric using Equation 4.1. Since, two metrics (i.e., complementarity and positive QA) have a positive impact and three metrics (i.e., dependencies,

Table 4. 4 Scores of design metrics for various tactics

Metric	Removal of Duplicates				Handling Missing Values				Feature Selection				Parallel Processing				Signature-based Detection				Anomaly-based Detection				Alert Ranking			
	KDD	DARPA	CIDDS	CICIDS2017	KDD	DARPA	CIDDS	CICIDS2017	KDD	DARPA	CIDDS	CICIDS2017	KDD	DARPA	CIDDS	CICIDS2017	KDD	DARPA	CIDDS	CICIDS2017	KDD	DARPA	CIDDS	CICIDS2017	KDD	DARPA	CIDDS	CICIDS2017
Dependencies	-0.63	-0.13	-.38	-0.63	-0.50	-0.75	-0.50	-0.75	-0.13	-0.63	-0.50	-0.50	0.00	-0.13	-0.75	-0.50	-0.38	-0.12	-0.50	-0.50	-0.25	-0.38	-0.38	-0.38	-0.63	-0.75	-1.00	-0.63
Complementarity	0.62	0.31	0.54	0.27	0.00	0.00	0.00	0.00	0.46	0.23	0.31	0.54	0.81	0.88	1.00	1.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.62	0.31	0.54	0.27
Constraints	-0.67	0.00	-0.33	-0.17	-0.50	-0.33	-1.00	-0.50	-0.50	-0.50	-0.83	-0.33	-0.33	0.00	0.00	-0.17	-0.17	0.00	0.00	-0.17	0.00	-0.17	-0.33	-0.67	-0.67	0.00	-0.33	-0.17
Positive QA	0.79	0.37	0.47	0.00	0.21	0.26	0.26	0.32	0.79	0.26	0.05	0.79	0.63	0.68	0.84	1.32	0.89	1.34	0.26	0.37	0.47	0.47	0.47	0.63	0.79	0.16	0.37	0.00
Negative QA	0.00	-0.29	0.00	-0.71	-0.71	-0.43	-0.43	-1.00	0.00	0.00	0.00	0.00	-0.29	-0.29	-0.29	-0.29	-0.29	-0.19	-0.29	-0.29	-0.14	-0.29	-0.43	-0.71	0.00	-0.29	0.00	-0.71
Net Utility	**0.11**	**0.27**	**0.30**	**-1.24**	**-1.50**	**-1.25**	**-1.67**	**-1.93**	**0.62**	**-0.63**	**-0.97**	**0.49**	**0.82**	**1.16**	**0.81**	**1.36**	**0.07**	**1.09**	**-0.52**	**-0.58**	**0.08**	**-0.35**	**-0.66**	**-1.12**	**0.11**	**-0.57**	**-0.43**	**-1.24**

constraints, and negative QA) have a negative impact, the range of weights is [-1,0] for metrics with negative impact and [0,1] for metrics with positive impact. After calculating the metrics' score, we use Equation 4.2 to calculate the utility score for each tactic as shown in Table 4.4.

4.6.2.3 Step 3: Workflow generation and reduction

In this step, we first use Equation 4.2 to determine the total number of workflows. For the seven considered tactics, five appears before and one (alert ranking) after the anomaly-based detection (Figure 4.10). Thereby $a = 5$ and $b = 1$ in Equation 4.2, which gives us a total number of **62** workflows. We then filter the 62 workflows through Algorithm 4.1 to discard workflows where a tactic does not have its complementary tactic present in the workflow. For example, the workflow T1→T2→T5→T6 is discarded as T1 and T2 depend upon T4, which is not present in the workflow. The filtering process reduces the number of potential workflows to **32**, which are presented in Table 4.5.

4.6.2.3 Step 4: Workflow selection

After obtaining the list of potential workflows, *TACTics* feeds the utility score of the tactics to Equation 4.3 to calculate the overall utility for each workflow. The utility scores for the workflows are shown in Table 4.5. For KDD, *WF-21* is the most optimal workflow with a utility score of 1.62. Similarly, *WF-22, WF-2, and WF-4* are determined as the optimal workflows for DARPA, CIDDS, and CICIDS2017 datasets with a utility score of 2.16, 1.11, and 1.85 respectively.

4.6.3 Case study evaluation

In this section, we implemented various designs (workflows) generated in Section 4.6.2.3 to evaluate the effectiveness of *TACTics* to answer the question: *Whether the workflow determined as most optimal at design time (using TACTics) is also the most optimal at runtime?*

4.6.3.1 Instrumentation and setup

We implemented the designed BDCA system using Apache Hadoop [50]. The Hadoop framework is configured on an OpenStack cloud cluster that consists of 16 slave nodes and 1 master node. Each slave node is configured with 2 GB RAM and 20 GB disk space while the master node is configured with 8 GB RAM and 80 GB disk space. Each node runs CentOS 6.4 operating system and JVM 1.7.0. The tactics are implemented using Java MapReduce paradigm. For *Removal of Duplicates* tactic, we used *NullWritable* class from Hadoop library that discards exactly alike records. For *Handling Missing Values* tactic, we used the mapper function to monitor each feature of each record and discard records with missing values. We used InfoGain [142] algorithm for implementing *Feature Selection* tactic. For

Table 4. 5 Overall utility score for each workflow

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Workflows	T4-T6	T1-T4-T6	T2-T4-T6	T3-T4-T6	T4-T5-T6	T4-T6-T7	T1-T2-T4-T6	T1-T3-T4-T6	T1-T4-T5-T6	T1-T4-T6-T7	T2-T3-T4-T6	T2-T4-T5-T6	T2-T4-T6-T7	T3-T4-T5-T6	T3-T4-T6-T7	T4-T5-T6-T7	T1-T2-T3-T4-T6	T1-T2-T4-T5-T6	T1-T2-T4-T6-T7	T1-T3-T4-T6-T7	T1-T3-T4-T5-T6	T1-T4-T5-T6-T7	T2-T3-T4-T5-T6	T2-T3-T4-T6-T7	T2-T4-T5-T6-T7	T3-T4-T5-T6-T7	T1-T2-T3-T4-T5-T6	T1-T2-T3-T4-T6-T7	T1-T2-T4-T5-T6-T7	T1-T3-T4-T5-T6-T7	T2-T3-T4-T5-T6-T7	T1-T2-T3-T4-T5-T6-T7
KDD	0.82	0.93	-0.68	1.44	0.89	0.9	-0.57	1.55	1	1.01	-0.06	-0.62	-0.6	1.51	1.52	0.97	0.05	-0.5	-0.49	1.61	1.62	1.08	0	0.02	-0.54	1.59	0.12	0.13	-0.42	1.61	0.08	0.2
DARPA	1.16	1.43	-0.09	0.53	2.25	0.8	0.18	0.8	2.52	1.07	-0.72	1	-0.44	1.62	0.17	1.89	-0.45	1.27	-0.17	0.44	1.89	2.16	0.37	-1.08	0.65	1.26	0.64	-0.81	0.92	1.53	0.01	0.28
CIDDS	0.81	1.11	-0.86	-0.17	0.28	0.14	-0.56	0.13	0.58	0.44	-1.83	-1.38	-1.52	-0.69	-0.83	-0.38	-1.53	-1.08	-1.22	-0.53	-0.39	-0.08	-2.35	-2.5	-2.04	-1.35	-2.05	-2.2	-1.74	-1.05	-3.02	-2.72
CICIDS2017	1.36	0.12	-0.57	1.85	0.78	0.24	-1.81	0.62	-0.46	-1	-0.08	-1.16	-1.7	1.27	0.73	-0.35	-1.32	-2.39	-2.94	-0.51	0.03	-1.59	-0.66	-1.2	-2.28	0.15	-1.9	-2.44	-3.52	-1.09	-1.79	-3.02

Signature-based Detection, we employed the model proposed in [177] that uses C4.5 for generating attack signatures. We used Apache Mahout [52] (a distributed ML library) for implementing the ML algorithms employed in the *Anomaly-based Detection* tactic. For *Alert Ranking*, we used the ranking model proposed in [178] that ranks alerts based on the number of bytes embodied in each alert.

We used the utility function (Equation 4.4) to calculate the Quality-of-Service (QoS) delivered by a workflow 'k' at runtime. The utility function weighs and sums the values for the quality metrics presented in Table 4.1. In Equation 4.4, $\omega \in [0,1]$ specifies the importance of each quality metric as assigned by the user. The quality metrics in the numerator of the Equation 4.4 are undesirable (i.e., the lower the value, the higher the quality) and the metrics in the denominator are desirable (i.e., the higher the value, the higher the quality). The utility function aims to minimize undesirable quality metrics and maximize desirable quality metrics. Hence, the workflow with the lowest QoS value is the most optimal at runtime. Since the value functions (e.g., Prediction Time (k)) originally end up in different units (e.g., seconds for prediction time), the value functions need to be normalized into the range [0,1]. We use Equation 4.5 and Equation 4.6 proposed in [179] to normalize value function for undesirable metrics (e.g., Training Time (k)) and value functions for desirable metrics (e.g., Accuracy (k)) respectively. In Equation 4.5 and Equation 4.6, $V_N(k)$ and $V_P(k)$ respectively denote value functions for undesirable and desirable quality metrics. Furthermore, V_{max} and V_{min} specifies the maximum and minimum value for a quality metric and V_k specifies the metric value for the workflow under consideration.

$$QoS(k) = \frac{\omega_1 \times Training\ Time\ (k) + \omega_2 \times Prediction\ Time\ (k) + \omega_3 \times False\ Positive\ Rate\ (k)}{\omega_4 \times Accuracy\ (k) + \omega_5 \times Detection\ Rate\ (k) + \omega_6 \times F1\ Score\ (k)} \quad (4.4)$$

$$V_N(k) = \begin{cases} \frac{V_{max} - V_k}{V_{max} - V_{min}}, if\ V_{max} - V_{min} \neq 0 \\ 1, \quad if\ V_{max} - V_{min} = 0 \end{cases} \quad (4.5)$$

$$V_P(k) = \begin{cases} \frac{V_k - V_{min}}{V_{max} - V_{min}}, if\ V_{max} - V_{min} \neq 0 \\ 1, \quad if\ V_{max} - V_{min} = 0 \end{cases} \quad (4.6)$$

4.6.3.2 Evaluation results

We implemented each of the 32 workflows for each of the four datasets and five ML algorithms. We then calculated the QoS delivered by each workflow using Equation 4.4.

TACTics' Accuracy: Figure 4.11 presents the results for workflows with the top five most optimal QoS for each of the 20 experimental cases (i.e., 4 datasets × 5 ML algorithms). The red bar in Figure 4.11 specifies the workflow determined as most optimal at design time. It is important to note that due to the design of utility function (Equation 4.4), the workflow with the lowest QoS value is the most optimal and vice versa. We measured the effectiveness of *TACTics* using *Top-1 Accuracy* and *Top-5 Accuracy* proposed in [180]. In our case, Top-1 accuracy means how often the workflow with the most optimal QoS at runtime is the one determined at design time. On the other hand, Top-5 accuracy measures how often any of the workflow with top 5 most optimal QoS (out of total 32) at runtime is the one determined as most optimal at design time. According to Top-5 measure, ***the accuracy of our approach is 80%*** (16/20). The accuracy of our approach is 45% (9/20) according to the Top-1 accuracy measure. This means in 9 out of 20 cases, the workflow found to be most optimal at runtime is the same as the one determined by *TACTics* at design time. With respect to the datasets, the top-5 accuracy is 60% (3/5) for KDD, 80% (4/5) for DARPA, 80% (4/5) for CIDDS, and 100% (5/5) for CICDIDS2017.

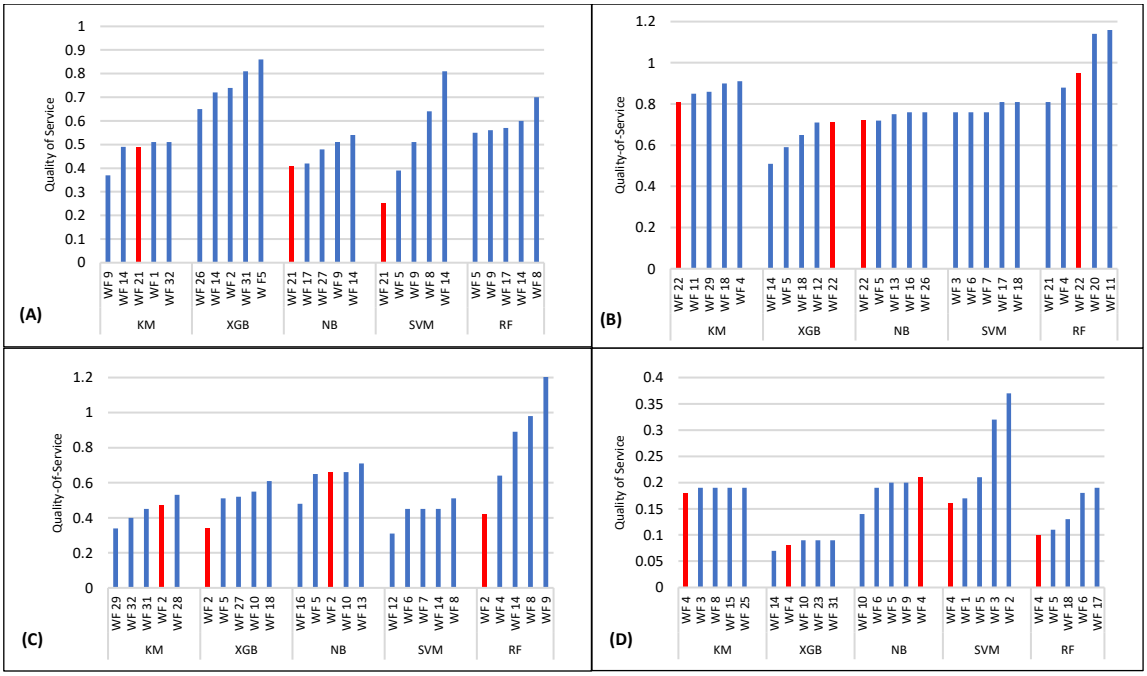


Figure 4. 11 QoS values of the top 5 most optimal workflows (WF) for (A) KDD (B) DARPA (C) CIDDS (D) CICIDS2017 datasets

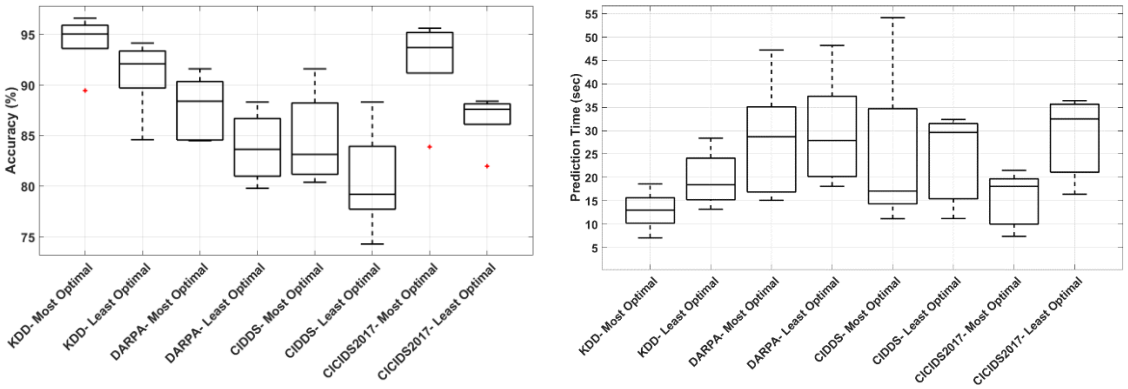


Figure 4. 12 Difference between Accuracy and Prediction time of most and least optimal workflows

Optimization: We also quantified the extent of optimization possible through efficient design using *TACTics* by comparing the QoS metrics (i.e., accuracy and prediction time) between the most optimal and least optimal workflow at runtime. The boxplots in Figure 4.12 shows the accuracy and prediction time (pertaining to the most and the least optimal workflows) averaged out for the five algorithms for the four datasets. For example, the average accuracy for the most optimal and least optimal workflows for KDD is around 95.07% and 93.62% respectively. By comparing the most and least optimal workflows, it is found that the *accuracy can be improved by up to 4.66%* and the *prediction time can be reduced by up to 20.56%* using *TACTics*. First, this result specifies that a loss of 4.66% in accuracy and 20.56% in prediction time is possible without using *TACTics*. Second, the result implies that in 45% of the cases (Top-1 measure), *TACTics* optimizes accuracy and prediction time on average by 4.66% and 20.56%. In 80% of the cases (Top-5 measure), the average optimization is around 3.49% (for accuracy) and 16.9% (for prediction time) as determined from the most optimal QoS values.

Weights' Accuracy: As mentioned in Section 4.5.3.2, in order to determine an optimal architecture, a software architect has to assign weights for the five design metrics associated with each tactic. Here, we evaluate how accurate are the weights we assigned based on the guidelines presented in Section 4.5.3.2. We evaluate the accuracy of the weights based on two methods: (i) comparison of the manually

assigned weights with the weights obtained from regression analysis and (ii) comparison of the manually assigned weights with the weights obtained from the net utility of individual tactic as determined from the study presented in Chapter 3.

In the first method, we performed Lasso Regression Analysis [181] to determine the relation between dependent variable and independent variable. In our case, the dependent variable is the overall utility of a workflow, which is the known variable as computed based on implementing each workflow. The independent variable is the weight i.e., the five weights associated with the five metrics for each tactic. The regression analysis aims to compute the unknown value of the weights based on the known value of the overall utility using Equation 4.7. In Equation 4.7, Utility (k) denotes the utility of workflow 'k' as determined from Equation 4.4, 'c' denotes the constant value of the design metric for a tactic as determined from Table 4.3, 'ω' specifies the unknown weight, and 'T' specifies the tactic which is equal to 1 if the tactic is present in the workflow else it is equal to 0. Moreover, 'n' specifies the number of tactic and 'm' specifies the number of design metric. The weights determined based on regression analysis are presented in Table 4.6. The net utility of each tactic computed based on manually assigned weights and the weights determined from regression analysis are shown in Table 4.7. The values of net utilities are normalized in the range [0,1] in order to make a comparison between the two types of utilities. We compute the total error and average error between the two type of utilities. The total error/dissimilarity between the two set of utilities is equal to 6.8. This means that the percentage error equals to 28.3% (6.8/24*100) where 24 is the maximum total error possible. In order words, 71.7% of the weights are correctly assigned based on the guidelines presented in Section 4.5.3.2.

Table 4. 6 Weights determined for each of the five design metrics from regression analysis

Tactic ID	Tactic	Dataset	Dependencies	Complementarity	Constraints	Positive QA	Negative QA
T1	Removal of Duplicates	KDD	-0.48	0.36	-0.59	0.67	-0.74
		DARPA	-0.45	0.38	-0.56	0.70	-0.72
		CIDDS	-1.00	0.00	-1.00	0.00	-1.00
		CICIDS2017	-0.79	0.14	-0.83	0.27	-0.89
T2	Handling Missing Values	KDD	-0.29	0.37	-0.43	0.84	-0.64
		DARPA	-0.25	0.37	-0.40	0.87	-0.62
		CIDDS	-0.11	0.37	-0.29	0.99	-0.55
		CICIDS2017	-0.24	0.37	-0.39	0.88	-0.61
T3	Feature Selection	KDD	-0.55	0.28	-0.79	0.58	-0.79
		DARPA	-0.40	0.44	-0.42	0.77	-0.68
		CIDDS	-0.49	0.33	-0.66	0.65	-0.75
		CICIDS2017	-0.53	0.30	-0.73	0.61	-0.78
T4	Parallel Processing	KDD	-0.38	0.81	-0.44	0.80	-0.69
		DARPA	-0.45	0.43	-0.55	0.71	-0.72
		CIDDS	-0.35	0.98	-0.40	0.83	-0.67
		CICIDS2017	-0.34	1.00	-0.39	0.84	-0.67
T5	Signature-based Detection	KDD	-0.51	0.37	-0.61	0.65	-0.78
		DARPA	-0.31	0.37	-0.45	0.82	-0.58
		CIDDS	-0.59	0.37	-0.68	0.58	-0.86
		CICIDS2017	-0.50	0.37	-0.61	0.65	-0.77
T7	Alert Ranking	KDD	0.02	0.37	0.00	1.00	0.00
		DARPA	-0.13	0.37	-0.18	0.90	-0.23
		CIDDS	-0.12	0.37	-0.17	0.91	-0.22
		CICIDS2017	-0.05	0.37	-0.08	0.95	-0.11

Table 4. 7 Comparison of tactic's utility obtained from manual weight assignment, regression analysis, and empirical findings based on tactic's impact

Tactic ID	Tactic	Dataset	Net Utility based on Manual Weights	Net Utility based on regression analysis	Net Utility based on quantification of tactic's impact
T1	Removal of Duplicates	KDD	0.662	0.535	0.381
		DARPA	0.441	0.560	0.126
		CIDDS	0.488	0.000	0.478
		CICIDS2017	0.226	0.214	0.016
T2	Handling Missing Values	KDD	0.139	0.676	0.280
		DARPA	0.222	0.706	0.135
		CIDDS	0.087	0.806	0.145
		CICIDS2017	0.000	0.713	0.441
T3	Feature Selection	KDD	0.477	0.428	0.288
		DARPA	0.421	0.638	0.035
		CIDDS	0.311	0.501	0.158
		CICIDS2017	0.785	0.456	0.519
T4	Parallel Processing	KDD	0.891	0.758	0.137
		DARPA	1.000	0.579	0.228
		CIDDS	0.886	0.836	0.322
		CICIDS2017	0.695	0.847	0.313
T5	Signature-based Detection	KDD	0.647	0.517	0.192
		DARPA	0.650	0.674	0.309
		CIDDS	0.456	0.441	0.292
		CICIDS2017	0.437	0.513	0.206
T7	Alert Ranking	KDD	0.652	1.000	0.319
		DARPA	0.511	0.861	0.000
		CIDDS	0.411	0.863	0.348
		CICIDS2017	0.262	0.934	0.332

In the second method, we first compute the net utility of each tactic based on the findings presented in Chapter 3, where we empirically quantify the impact of each tactic on four metrics i.e., accuracy, false positive rate, training time, and prediction time. Based on these metrics, we compute the net utility 'Utility (T)' of each tactic using Equation 4.8, where impact on M_i specifies the percentage increase/decrease in the value of the metric with the incorporation of the tactic. For example, the incorporation of Removal of Duplicates tactic increases accuracy of the system by 3.61% with the KDD dataset. The weights ω_i denotes the importance of the metric 'i', which we kept equal to 1 for all metrics in order to specify equal weight for all four metrics. Table 4.7 compares the net utility of each tactic calculated based on manual assignment of weights and empirical findings from Chapter 3. The total error between the two set of values is equal to 7.17, which means the percentage error equal to 29.8% (7.17/24). According to this method, 70.2% of the weights we assigned are correct.

$$Utility\ (k)\ =\ \sum_{n\ =\ 1}^{7}\sum_{m\ =1}^{5} c_n^m\ \times\ (\omega_n^m\ \times\ T_n\ ) \tag{4.7}$$

$$Utility\ (T)\ =\ \sum_{i\ =\ 1}^{4} \omega_i\ \times\ Impact\ of\ M_i \tag{4.8}$$

4.7 Threats to Validity

Internal Validity: The weighting process (in step 2 of *TACTics*) is limiting as it primarily depends upon an architect's experience. Whilst the limitation is common to most of the design approaches (e.g., [182, 183]), we provide and evaluate general guidelines to encourage a more formal treatment of the weighting process. ***Construct Validity:*** The utility function for assessing the QoS does not encompass all aspects (e.g., cost, scalability, and reliability) of a system, which poses a threat to the validity of our findings. ***External Validity:*** In our experiments, we used only four datasets and five ML algorithms. Whilst these datasets and ML algorithms are selected based on their widespread use in the domain of BDCA, the selection of a subset from a larger pool of datasets and ML algorithms does pose a threat to the generalization of the findings reported in this chapter. The tool support for the implementation of the tactics in guidance models is not limited to what is included in the guidance models. We believe a large body of such tools exists (e.g., Hadoop Ecosystem [50]) and can be added to the guidance models.

4.8 Related Work

Guidance models attracted a lot of attention of the software engineering community. MacLean et al., [184] introduced Questions-Option-Criteria (QOC) approach, which provides a basis for guidance models. The QOC approach maps questions (problem space) to the options (solutions space) and uses the criteria to select the potential options. Afterward, several works are undertaken to propose guidance models in various domains such as cyber-foraging [168], microservices [169], and cloud computing [185]. Motivated by such efforts and utmost need for design guidance [1], ours is the first effort to propose guidance models for designing BDCA systems. Several design approaches (e.g., [166, 182, 183, 186, 187]) have been developed to assist software architects. Sabry et al., [166] conducted a survey with architects to quantify the importance of various quality attributes and satisfying them using well-known tactics such as authentication and timeout. Soliman et al., [186] proposes a design approach for making decisions related to the selection of technologies based on their impacts on a system's architecture. Al-Naeem et al., [183] proposed a systematic and quality-driven approach that uses Analytic Hierarchy Process (AHP) to explore and quantitatively compare various technological options (such as programming languages and databases) for designing a distributed software system. Svahnberg et al., [182] also used AHP method to compare a rather small number of architecture candidates based on quality attribute preference. Whilst the previous works are in different domains and differ in design approach, the work that relates more closely to ours is the work of Fahmideh et al., [187] which propose various architectures for a big data system. Unlike our approach, their work is based on design obstacles and uncertainties to propose an architecture. Moreover, their approach is not directly applicable to BDCA systems.

4.9 Chapter Summary

In this chapter, we propose *TACTics*, a design approach for the systematic and optimal design of a BDCA system. *TACTics* consists of *guidance models* and a *heuristic-based design*. The *guidance model* provides guidance on various design considerations (e.g., implications on quality goals), which are then used by the *heuristic-based design* to compare and determine an optimal design that best satisfies the quality goals and system constraints. We evaluate *TACTics* with a case study on designing a BDCA system for intrusion detection. The results of our case study show that in 80% of the cases the architecture recommended by *TACTics* at design time is included in the top five most optimal architectures at runtime.

The work reported in this chapter has several implications for practitioners/architects. The design approach, *TACTics*, does not attempt to replace an architect rather helps an architect of a BDCA system by (a) *Revealing design knowledge:* the guidance models, first part of *TACTics*, reveal the list of functional and non-functional requirements, tactics used to address the requirements, the impact of the tactics on quality attributes, the relationship among the tactics, the constraints associated with the tactics, and the implementation options for the tactics. Such design knowledge increases the understanding of the architect and helps in making well-informed design decisions (b) *Quantifying and justifying design decisions: TACTics* enables an architect to quantify, through the assignment of weights, the impact of each tactic on various design metrics (e.g. positive impact on QA) early at the design stage. Such quantification provides a formal way to justify and rationalize design decisions (c) *Systematizing design process:* one of the main benefits of *TACTics* is that it forces an architect to systematically consider all candidate architectures and then, through quantitative assessment, select an architecture that best satisfies quality goals and respect system constraints (d) *Enabling design traceability: TACTics* provides a more formal way of recording relationships among the elements of the engineering process especially the relationship between requirements and design. Understanding and recording the relationships among the elements help an architect in critical cross-linking, justifying decisions, and auditability (e) *Unfolding significance of design choices:* our design approach and its evaluation helps in realizing the significance of making the right design choices. For instance, Section 4.2.3 and Section 4.6.3.2 reveals that architectural configuration significantly impacts various quality attributes such as accuracy and prediction time. Such findings serve as a motivation for architects to critically assess and compare alternative architectures for designing BDCA systems.

Chapter 5

Architecture-driven Adaptation of Big Data Cyber Security Analytics Systems

Related publication:

This chapter is based on our paper published in *International Conference on Software Architecture (ICSA 2019)* "An Architecture-driven Adaptation Approach for Big Data Cyber Security Analytics"[5].

Chapter 2 reveals that *accuracy* and *response time* are the two most important quality concerns for BDCA systems. However, the frequent changes in the operating environment of a BDCA system (such as quality and quantity of security event data) significantly impact these qualities. In this chapter, we first study the impact of such environmental changes. We then present *ADABTics*, an architecture-driven adaptation approach that (re)composes the system at runtime with a set of components to ensure optimal *accuracy* and *response time*. We evaluate our approach both in a single node and multi-node settings using a Hadoop-based BDCA system and different adaptation scenarios. The research presented in this chapter highlights the importance of runtime adaptation for BDCA systems and demonstrates the effectiveness of our architecture-driven adaptation approach for BDCA systems.

5.1 Introduction

In addition to commonly known factors (e.g., algorithm efficiency, hardware failures, and parameter configuration) that impact *accuracy* and *response time*, the software architecture of a BDCA system plays a significant role in accurately and quickly detecting attacks [1, 188, 189]. As observed in Chapter 2, researchers and practitioners use various tactics to achieve different quality attributes (e.g., accuracy, scalability, and reliability) in a BDCA system [1]. For instance, an anomaly-based detector (which detects attacks based on deviation from normal behaviour) is integrated with a signature-based detector (which detects attacks based on known attack patterns) to boost *accuracy* [158]. A tactic for boosting the *response time* is the selection of features from the security data to reduce data size [142].

The effectiveness of these tactics varies with changes in an operating environment. In our context, operating environment means the enterprise network where a BDCA system is deployed for detecting attacks. Among others, the most important changes are the changes in the quality and quantity (e.g., change in the number of features in data) and velocity of security event data ingested into a BDCA system [92]. Any such change has a direct impact on the effectiveness of the employed tactics and ultimately the *accuracy* and *response time* of the system. For example, feature selection tactic can help reduce response time in an environment, which generates security data with a large number of features. Otherwise, the tactic only negatively impacts the response time (Section 5.2.2). Accuracy and

response time are not impacted equally with changes. For example, a BDCA system, having fixed processing power (e.g., number of CPUs), subjected to an increase in data influx rate will make an entire compromise on *response time* while keeping the *accuracy* largely intact (Section 5.2.3). This leads to one quality (i.e., *accuracy*) being optimized and another quality (i.e., *response time*) being totally compromised. Furthermore, some tactics expose a trade-off situation between accuracy and response time. To illustrate the trade-off, consider the case of integrating an anomaly-based detector with a signature-based detector in a BDCA system. The integration enables two-phase processing – security data is first processed by the signature-based detector followed by the anomaly-based detector, which makes sure that most of the malicious activities are detected [158]. However, such an increase in accuracy comes at the cost of increase in response time due to two-phase processing (Section 5.2.1). This raises the concern whether such tactics should be employed. Given the dynamic nature of a BDCA system's operating environment [189] and the aforementioned sensitivities to the changes in operating environment pose the problem - "***How to enable a BDCA system to ensure optimal accuracy and response time in the face of changes in the operating environment?***".

In this chapter, we present *ADABTics*, an **A**rchitecture-**D**riven **A**daptation approach for **B**ig data Cyber security Analy**Tics**, which combines *component-based architecture* [190, 191] with *architecture-driven adaptation* [192, 193] to ensure optimal accuracy and response time. By architecture-driven adaptation, we mean the ability of a BDCA system to modify its structure in terms of adding and removing components at runtime in response to changes in the operating environment. It is important to note that *TACTics* presented in Chapter 4 aims to determine optimal architecture at design time while *ADABTics* aims to optimize accuracy and response time by determining optimal architecture at runtime in accordance with the changes in the operating environment. The fundamental differences among the approaches presented in the four chapters (i.e., *TACTics* in Chapter 4, *ADABTics* in Chapter 5, *QuickAdapt* in Chapter 6, and *SCALER* in Chapter 7) are presented in Table 5.1.

ADABTics views and subsequently composes a BDCA system as a set of independent components. Each component implements a tactic (e.g., feature selection) devised for boosting accuracy or response time. In addition to other advantages (e.g., code reuse, maintainability, and independent development), organizing a BDCA system as a set of components renders the flexibility to add and remove components from the available pool of components at runtime [190]. Such flexibility is exploited by the *ADABTics*' adaptation mechanism, which monitors the accuracy, response time, and properties of the operating environment (i.e., data quality and influx speed) of a BDCA system. Upon detection of deviation from the expected values set by a user (e.g., accuracy falls below 80%), adaptation is triggered, which re-composes the architecture of the system constituting a different set of components. The adaptation ensures that the system, now composed of a new set of components, delivers optimal accuracy and response time. We have implemented *ADABTics* on Hadoop [50] for

Table 5. 1 Differences among approaches presented in chapter 4, 5, 6, and 7.

Chapter	Approach	Application	Quality Attributes	Description
4	*TACTics*	Design time	Accuracy and response time	Determining architecture as a selection of tactics based on user assigned weights
5	*ADABTics*	Runtime	Accuracy and response time	Determining architecture based on changes in the operating environment of the system
6	*QuickAdapt*	Runtime	Accuracy and response time	Determining architecture based on correlating security data with components of the system
7	*SCALER*	Runtime	Scalability	Determining big data framework configuration based on changes in the operating environment

evaluation in three *execution modes* using four well-known security datasets. Our evaluation reveals that on average *ADABTics* improves a BDCA system's accuracy and response time by 6.06% and 23.7% respectively.

Contributions: In this chapter, we make the following contributions.

1. Demonstrate the impact of changes in the operating environment on the accuracy and response time of a BDCA system (Section 5.2)
2. Propose a component-based architecture for BDCA (first part of *ADABTics*) (Section 5.3.1)
3. Introduce an architecture-driven adaptation approach for BDCA (second part of *ADABTics*) (Section 5.3.2)
4. Implement and evaluate the effectiveness of *ADABTics* through comprehensive experiments (Section 5.5)

Chapter Organization: Section 5.2 demonstrates the impact of changes in the operating environment to motivate the undertaken work. Section 5.3 describes our adaptation approach, which is followed by implementation details of the approach in Section 5.4. Section 5.5 presents the evaluation results. Section 5.6 discusses the results. Section 5.7 outlines the threats to the validity of the findings presented in this chapter. Related work is presented in Section 5.8. Section 5.9 concludes the chapter.

5.2 Motivation

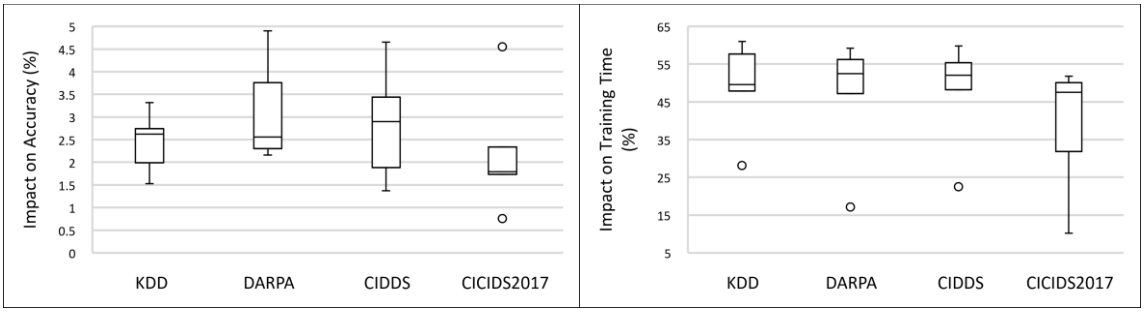
In this section, we illustrate the accuracy-response time trade-off and expose the impact of environmental changes on the accuracy and response time of a BDCA system.

5.2.1 Accuracy-Response time trade-off

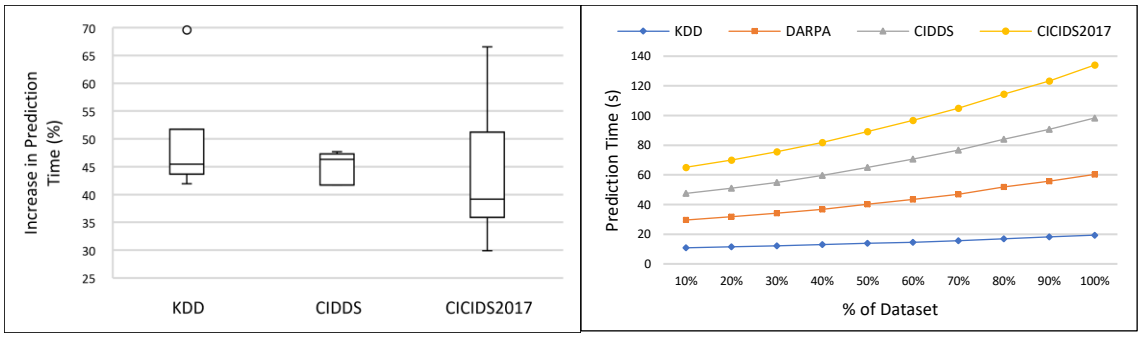
Accuracy and response time pose a trade-off situation (i.e., improving one compromises another) at several design decisions in a BDCA system. Such decisions are characterized by the incorporation of architectural components. For example, integrating a Signature-based Detection component with an Anomaly-based Detection component improves accuracy but at the cost of increased response time. To exemplify such trade-offs, we investigated the impact of two important components (i.e., Signature-based Detection and Alert Ranking (AR)) on accuracy and response time in a Hadoop-based BDCA system. The metrices used for measuring accuracy and response time are presented in Table 4.1 and the implementation details of the experiment are available in Section 5.4. The boxplots in Figure 5.1 shows the difference between the accuracy and response time of the system with and without the components (i.e., Signature-based Detection and Alert Ranking). Figure 5.1a and Figure 5.1b illustrates that on average the use of Signature-based Detection component increases accuracy by 2.31% but at the cost of 46.06% increase in training time. Our qualitative analysis of the ranked alerts indicates that the alerts ranked on top are more severe, hence, the use of Alert Ranking component improves usability and subsequently accuracy. However, Figure 5.1c shows that the component increases the prediction time by 46.2% on average. Given that a BDCA system is architected using a variety of components [1], these results highlight the underlying trade-offs and point towards the challenging decision about the selection of components. For example, whether or not Alert Ranking component should be included in the BDCA architecture.

5.2.2 Sensitivity of architectural components to data variety

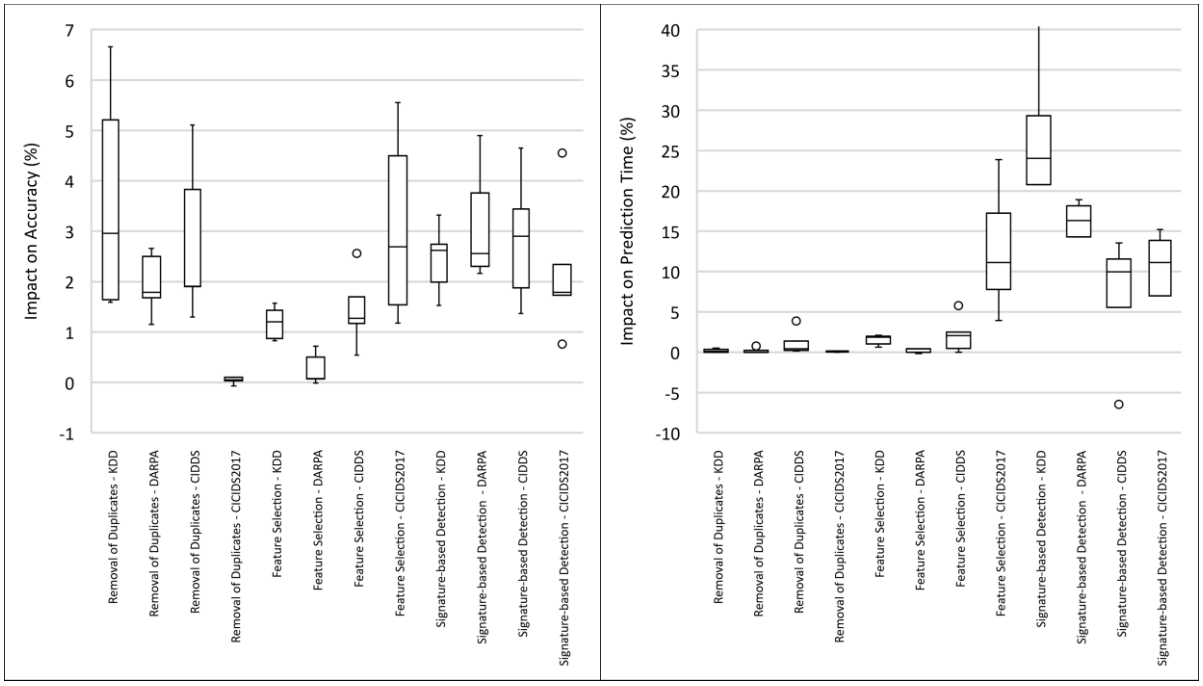
The characteristics of the security data (e.g., number of features in each network connection or amount of incorrect values in the collected security data) often changes in the underlying network [92, 189]. Such changes impact the performance of various components in the system. To illustrate the impact, we investigated the impact (in terms of accuracy and response time) of change in data quality on various components. Figure 5.1e shows the impact on accuracy observed for three components (i.e., Removal of Duplicates, Feature Selection, and Signature-based Detection) for different security



(a) Increase in accuracy with inclusion of Signature-based Detection **(b)** Increase in training time with inclusion of Signature-based Detection



(c) Increase in prediction time with inclusion of Alert Ranking **(d)** Impact of data size on prediction time



(e) Impact of inclusion of various components on accuracy **(f)** Impact of inclusion of various components on prediction time

Figure 5. 1 Impact of changes in operating environment on accuracy and response time

datasets. The impact is measured by executing the experiment on the system with and without the component. It is evident from Figure 5.1e that the impact (or contribution) of the components varies with the change in data quality. For instance, the accuracy for KDD [55] improves by up to 6.7% with the incorporation of the Removal of Duplicates component while the same component has negligible impact for CICIDS2017 [58]. The same variation is observed for prediction time (Figure 5.1f). The use of Feature Selection component reduces the average prediction time by 12.7% for CICIDS2017 but achieves a reduction of only 1.51% for KDD. Similarly, Signature-based Detection component reduces the prediction time by 24% for KDD but only achieves a reduction of 10% for CICIDS2017. From these findings, it can be deducted that the various components in a BDCA system are highly sensitive to quality of security data. Hence, the components should be operationalized only when the operating environment (e.g., data quality) is as such that the component can play a significant role. For instance, operationalizing Removal of Duplicates component for CICIDS2017 will only be counterproductive as evident from Figure 5.1f.

5.2.3 Sensitivity of architectural components to data velocity

The velocity of security data in the underlying network infrastructure is quite dynamic [92, 189]. For example, the network of a bank is very active in working hours as compared to non-working hours. Thereby, data is generated at a higher speed in working hours. The response time of a BDCA system (having fixed computational capacity) is likely to decrease with the increase in the data speed. To better highlight this relation, we conducted an experiment where a BDCA system is subjected to varying size of security data to be processed at a given time. In each execution, the system is processing a subset (e.g., 10%, 20%, and so on) of the total dataset. Our results show that the amount of data (to be processed at a given time) does not have any significant impact on the accuracy. On the other hand, Figure 5.1d illustrates a constant increase in the prediction time as the size of the data increases. For instance, the prediction time for KDD increases from 8 sec to 20 sec as the size of data increases from 10% to 100%. The findings of this experiment enable us to conclude that as the data speed (and subsequently data size) increases, the BDCA system is keeping the accuracy largely intact and making the entire compromise on the prediction time, which is not an effective approach. A delay in response to an attack can have detrimental consequences [1].

5.3 *ADABTics* – An Adaptation Approach for BDCA

We present our approach, *ADABTics*, that enables a BDCA system to ensure optimal accuracy and response time in the face of changes in the operating enviornment. *ADABTics* consists of two parts – component-based architecture and adaptation approach.

5.3.1 Component-based architecture for BDCA

In this section, we first present the meta-model followed by the component-based BDCA architecture.

5.3.1.1 Metamodel for BDCA architecture

The BDCA meta-model characterizes the elements of a BDCA architecture and the relationship among the elements. The meta-model consists of four architectural elements shown in Figure 5.2. **Component:** A software component is a unit of encapsulated behavior that is used to organize operations and data *[190]*. **Interface:** An interface is a collection of operations that are used to access a component. The interfaces in our model are defined in terms of ports, which are points of connection between a

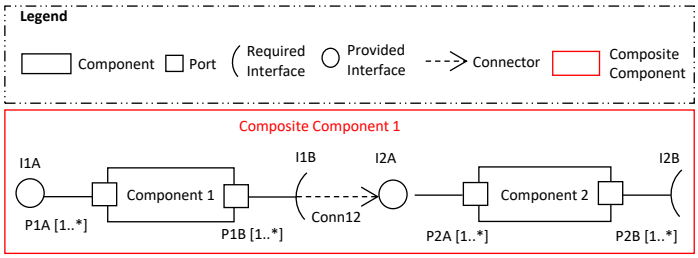


Figure 5. 2 Metamodel for BDCA architecture

component and its environment. For instance, Figure 5.2 shows that Component 1 has two types of ports (i.e., P1A [1..*] and P1B [1..*]) with associated interfaces (i.e., I1A and I1B). **Connector:** A connector is a runtime communication link between two components *[194]*. Each connecter is specified by its name and the interfaces it connects. **Composition:** In our model, the entire BDCA system is modeled as a group of composite components – each group termed as a module, which is composed of other components. Some modules, depending on the complexity, are further divided into phases where each phase is composed of several components. Our architecture also considers any external tool (e.g., dashboard) as a component to which the rest of the system is connected as per the metamodel presented in Figure 5.2.

5.3.1.2 BDCA component-based architecture

The component-based architecture of a BDCA system is presented in Figure 5.3 as managed system. The component-based architecture presented in Figure 5.3 is developed based on the BDCA reference architecture presented in Chapter 2 and guidance models presented in Chapter 4. We first present the architecture overview followed by the description of various features. *Overview:* The architecture is organized into layers, modules, phases, and components with layers being at the highest abstraction level and components at the lowest. There are two layers – *Security Analytics Layer* that collects, prepares, and analyzes the security data and *Big Data Support Layer* that supports the storage and processing of the large volumes of heterogenous natured security data. Each layer is composed of several modules with *Security Analytics Layer* being composed of three modules (i.e., Data Collection, Data Analytics, and Visualization) and *Big Data Support Layer* being composed of two modules (i.e., Storage Support and Processing Support). Each module embodies several components except for Data Analytics module, whose complexity first warrants a categorization into different phases pertaining to the data lifecycle.

Security Analytics Layer

This layer implements the key features responsible for detecting attacks. It is composed of the following three modules.

Data Collection: This module collects security data from data sources (e.g., Packet data, NetFlow data, IDS log data, firewall logs, email servers, operating system logs, and application servers). This way external data sources are connected to the BDCA system. Data Collection module uses different tools (e.g., Wireshark and nmap) for collecting the data.

Data Analytics: The data analytics module is responsible for analyzing the collected security data to extract the desired insight (i.e., information about malicious cyber activities). The module consists of five phases. Data Engineering: This phase engineers or preprocess the security data to enable it to be used efficiently in the subsequent phases. The phase incorporates the following components. The *Removal of duplicates* component removes duplicate records from the security data. This is important

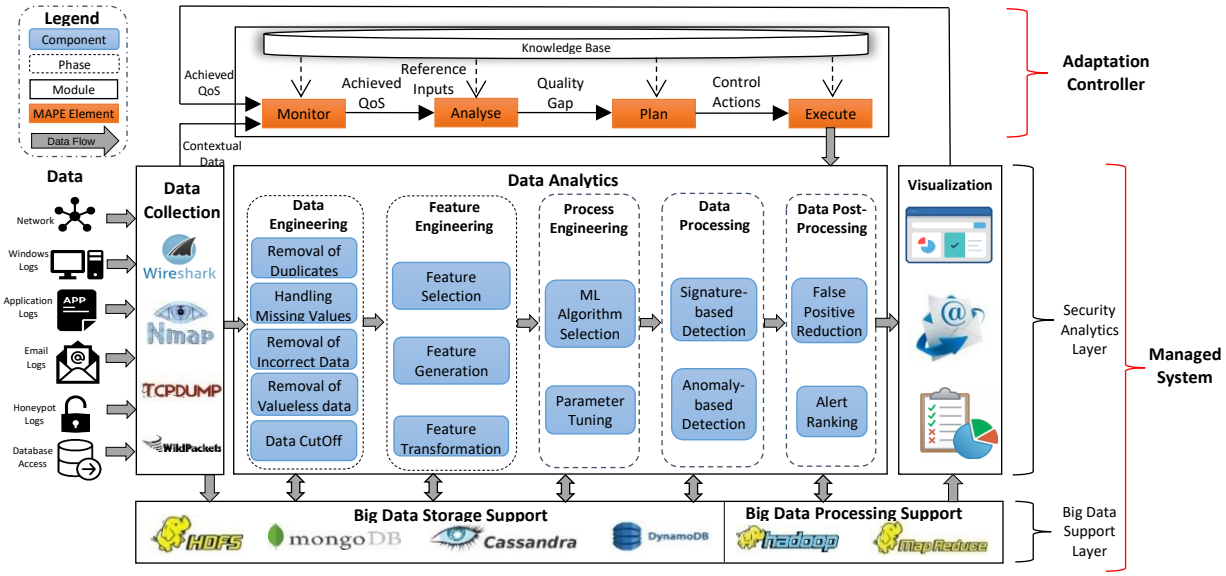


Figure 5. 3 BDCA component-based architecture

because the existence of duplicate records makes the Machine Learning (ML) model (employed in the Anomaly-based Detection component) biased towards more frequent records and prevents it from learning rare behaviors that pertains to more dangerous attacks. The *Handling missing values* component asses the number of records in the collected security data that has missing values. Based on the statistical assessment, this component either deletes the records with the missing values or opt for data imputation to fill the missing values. The *Removal of incorrect data* component filters the collected data to remove records with incorrect values. For example, a negative value (e.g., -20) for number of bytes in a network connection is considered as an incorrect value. The *Removal of valueless data* component removes the data that does not contribute to attack detection. For example, a network sniffer collects zero-byte data, which is used for handshaking in TCP/IP connection. Such data does not have any relevance for security analytics; hence, it should be discarded. The *Data cutOff* component imposes a customizable limit on the data to be included for a connection or process. For instance, the component can impose a cutoff limit of 15 KB for each network connection, which ensures to only retain the first 15 KB data for each connection. *Feature Engineering:* This phase is responsible for selecting, generating, and/or transforming features from the engineered data. Given that each record in security data is composed of several features (e.g., 41 features in KDD and 82 features in CICIDS2017) and not all of them contribute to attack detection, the selection of specific features for subsequent processing is crucial for achieving accurate and speedy results. This phase consists of four components. The *feature selection* component leverages intelligent algorithms (e.g., InfoGain [142]) to select the most relevant features. The *Feature generation* component creates new features from the available features with the aim of accuracy improvement. To help the ML model discriminate more clearly among various classes (e.g., benign and malicious record), the *Feature transformation* component transform the features into a different space using techniques such as normalization of features into a range of 0 to 1. *Process Engineering:* This phase consists of components responsible for taking various measures to ensure effective data processing in the subsequent step as shown in Figure 5.3. In our architecture, the phase involves components – (1) *ML algorithm selection* component selects the most suitable ML algorithm based on applying and validating available algorithms (2) *Parameter tuning* component handles the tuning of various parameters (e.g., value of K for K-means) required in data processing phase. *Data Processing:* This phase processes the prepared security data in parallel fashion to extract insight that signals towards a potential attack. Based on the processing technique, BDCA can be

categorized into (i) Signature-based detection and (ii) Anomaly-based detection [92]. Accordingly, this phase consists of two components. The *Signature-based detection* component detects attacks based on already known attack patterns while *Anomaly-based detection* component uses ML algorithms to detect attacks based on deviation from already learned normal behavior. *Data Post-processing:* This phase processes the analyzed results to remove (any) false alarms and facilitate the user in responding to the detected attacks. The phase consists of two components – *False positive reduction* component leverages vulnerability signatures to reduce the false positive rate and *Alert ranking* component ranks the generated alerts based on their dangerousness and severity [157] to facilitate a user in responding to more dangerous attacks on priority basis.

Visualization: This module consists of tools (e.g., dashboard, email notifier, and reports) that are used to communicate the results of security analytics to the user.

Big Data Support Layer

This layer leverages big data technologies to support the *Security Analytics layer* by managing all the issues related to data storage and data processing distribution. The layer consists of two modules. The *Big Data Storage Support* module manages the back and forth storage of the security data after being processed in each module and phase of the *Security Analytics layer*. The module uses different data storage tools (e.g., HDFS, MongoDB, or Cassandra) to support the required data storage. The *Big Data Processing Support* module provides the processing framework for distributed processing of the data. To support such processing, the module uses big data framework (i.e., Hadoop) and its programming paradigm (i.e., MapReduce).

5.3.2 Adaptation Approach

We now present our adaptation approach that leverages the component-based BDCA architecture presented in the previous section.

Adaptation Goal: Our adaptation goal specifies the primary reason as to why a BDCA system should be self-adaptive. As revealed in Section 5.2, the accuracy and response time, collectively measured as the Quality-of-Service (QoS), delivered by a BDCA system is sensitive to the changes in the operating environment. This sensitivity drives our adaptation goal, which is to "*enable a BDCA system to ensure optimal QoS in the face of changes in the operating environment*". Given that QoS encompasses a multitude of qualities (e.g., response time, scalability, and reliability), our focus is only on accuracy (measured in terms of attack detection rate, false positive rate, accuracy, and F1 score) and response time (measured in terms of prediction time). We attempt to achieve the adaptation goal by employing a MAPE-K loop for monitoring, analysing, planning, and executing the (re)composition of the BDCA architecture based on changes in the operating environment of the system.

System Architecture: The system consists of two subsystems –*managed system* and *adaptation controller*. The different aspects of the managed system (i.e., BDCA system) have already been reported in Section 5.3.1. The adaptation controller follows the structure of the MAPE-K loop [195], which consists of five main elements. *Monitor* collects the information related to the QoS delivered by the managed system. As shown in Figure 5.3, the QoS delivered by the managed system is directly monitored through the *monitor* element. The information collected by *monitor* is analyzed through *analyzer*, which compares it with the reference inputs and determines if the *quality gap* has crossed the *threshold*. In case of quality gap crossing the threshold (indicating that the system requires adaptation), the *analyzer* triggers the *planner* element. The *planner* computes the control actions that suggest a new configuration for the composition of the managed system as mentioned previously. The *executor* recomposes the structure

of the managed system according to the new configuration. The *knowledge base* maintains the data of the managed system (e.g., QoS constraints), operating environment (e.g., data influx rate), and data shared among the four MAPE elements (e.g., reference inputs).

Reference Inputs: The reference inputs specify the concrete target/state to be achieved in the BDCA system by our adaptation approach. Like most research on self-adaptation, the reference input in our approach is the *ideal QoS* that a BDCA system aims to achieve through the adaptation approach. Unlike Chapter 4, the utility function (Equation 5.1) used for calculating the QoS does not include training time. This is because the QoS in Chapter 4 is calculated at design time while the QoS in this chapter is calculated at runtime, where an already developed model is employed in the system. Equation 5.1 normalizes and sums the values for desired qualities. In Equation 5.1, *QoS(k)* is the obtained QoS at k^{th} iteration (configuration) and ω denotes the weight (specifying the importance) for a quality (e.g., detection rate) assigned by a user. Similar to Chapter 4, the value functions (e.g., *False Positive Rate(k)*) are scaled using Equation 5.2 and Equation 5.3 [179] before calculating *QoS(k)*. In Equation 5.2 and Equation 5.3, $V_N(k)$ and $V_P(k)$ denotes the value function. Some quality attributes have a negative impact i.e., the higher the value, the lower the quality (e.g., false positive rate) while some quality attributes have a positive impact i.e., the higher the value, the higher the quality (e.g., detection rate). Equation 5.2 is used to scale qualities having a negative impact and Equation 5.3 is used to scale qualities having a positive impact.

$$QoS(k) = \frac{\omega_1 \times Prediction\ Time\ (k) + \omega_2 \times False\ Positive\ Rate\ (k)}{\omega_3 \times Accuracy\ (k) + \omega_4 \times Detection\ Rate\ (k) + \omega_5 \times F1\ Score\ (k)} \tag{5.1}$$

$$V_N(k) = \begin{cases} \frac{V_{max} - V_k}{V_{max} - V_{min}}, if\ V_{max} - V_{min} \neq 0 \\ 1, \quad if\ V_{max} - V_{min} = 0 \end{cases} \tag{5.2}$$

$$V_P(k) = \begin{cases} \frac{V_k - V_{min}}{V_{max} - V_{min}}, if\ V_{max} - V_{min} \neq 0 \\ 1, \quad if\ V_{max} - V_{min} = 0 \end{cases} \tag{5.3}$$

Measured Outputs: These are the set of values that are measured and compared with the reference inputs to evaluate whether the managed system has achieved the desired state. In our approach, we use *achieved QoS* as the measured output. The *achieved QoS* specifies the QoS achieved after the system has adopted to the changes. The *achieved QoS* is measured using Equation 5.1. The difference between *ideal QoS* and *achieved QoS* is termed as *quality gap*. We define a *threshold* for the *quality gap*. If the *quality gap* is within the threshold, the system is delivering satisfied QoS and vice versa.

Control Actions: These are the actions undertaken to make the managed system adapt to the changes in the operating environment. With the introduction of any such change, the *quality gap* deviates from the already defined *threshold* range. Thereby, the control actions aim to bring the *quality gap* within the *threshold* range. In our approach, the control action is the re-composition of the architecture of a BDCA system. Each time a QoS violation is detected (i.e., *quality gap* exceeds *threshold* range), the control action is triggered, which computes the QoS (i.e., *achieved QoS*) delivered by different composition of the system's architecture. The compositions are specified by **A**rchitectural **C**onfigurations (AC), shortly referred as configuration, shown in Table 5.2. An architectural configuration underlines the components included in the architecture. A total of six components are considered in the configuration space, whose combinations forms a configuration. These six components are: (C1) Removal of duplicates (C2) Handling missing values (C3) Feature selection (C4) Signature-based detection (C5) Anomaly-based detection and C6) Alert ranking. For example, AC-2 consists of two components (i.e., Removal of Duplicates and Anomaly-based Detection) and AC-8 consists of three components (i.e., Removal of Duplicates, Feature Selection, and Anomaly-based Detection). The algorithm for the

Table 5. 2 Architectural configurations considered in the study

ID	Configuration	ID	Configuration	ID	Configuration	ID	Configuration
1	C5	9	C1-C4-C5	17	C1-C2-C3-C5	25	C2-C4-C5-C6
2	C1-C5	10	C1-C5-C6	18	C1-C2-C4-C5	26	C3-C4-C5-C6
3	C2-C5	11	C2-C3-C5	19	C1-C2-C5-C6	27	C1-C2-C3-C4-C5
4	C3-C5	12	C2-C4-C5	20	C1-C3-C5-C6	28	C1-C2-C4-C5-C6
5	C4-C5	13	C2-C5-C6	21	C1-C3-C4-C5	29	C1-C3-C4-C5-C6
6	C5-C6	14	C3-C4-C5	22	C1-C4-C5-C6	30	C1-C2-C3-C5-C6
7	C1-C2-C5	15	C3-C5-C6	23	C2-C3-C4-C5	31	C2-C3-C4-C5-C6
8	C1-C3-C5	16	C4-C5-C6	24	C2-C3-C5-C6	32	C1-C2-C3-C4-C5-C6

Algorithm. 5.1. Configuration Selection Algorithm

Input: QoS_{idl} = ideal QoS
$QoS_{thrshold}$ = QoS threshold
AC = {AC_1, AC_2, …., AC_N} // *Set of possible architectural configurations*
C = {C_1, C_2, …., C_Z} // *Set of user-defined constraints*
I_{max} = maximum number of iterations
S_{cond} = {QoS Satisfaction, User-defined Limit, Most Optimal QoS} // *user chooses stopping condition*

Output: AC_{opt} = Optimal architectural configuration

Steps:

```
1.  ACopt = AC1
2.  QoSgap = |Qidl – F(AC1)|
3.  If Scond = most optimal configuration
4.  |       Imax = N
5   End
6.  For i = 1 to Imax
7.  |       Calculate Utility Function QoS (ACi)
8.  |       QoSgap (ACi) = |QoSidl – QoS(ACi)|
9.  |       If QoSgap (ACi) > QoSthrshold
10. |       |     Jump to Next Iteration of ith For loop
11  |       End
12. |       Else
13. |       |     For j=1 to Z
14. |       |     |     If (ACi violates any constraint)
15. |       |     |     |     Terminate the jth For loop
16  |       |     |     End
17. |       |     |     Else
18. |       |     |     |     If (Scond = threshold entry)
19. |       |     |     |     |     ACopt = ACi
20. |       |     |     |     |     Terminate the process
21  |       |     |     |     End
22  |       |     |     |     Else
23. |       |     |     |     |     QoSgap = QoSgap (ACi)
24. |       |     |     |     |     ACopt = ACi
25  |       |     |     |     End
26  |       |     |     End
27  |       |     End
28  |       End
29  End
```

// *The final value of AC_{opt} is the optimal configuration selected for re-composition*

selection and ultimately recomposing the architecture of the system is shown as Algorithm 5.1. Upon activation of adaptation trigger, the algorithm starts calculating the *achieved QoS* for different architectural configurations (line 6 and 7) and discards configurations with *achieved QoS* greater than the *threshold* (line 8-10). There are three conditions (specified by S_{cond}) that stop and recompose the architecture – (i) *QoS satisfaction:* as soon as the QoS for a configuration is found within the *threshold* range, the process is stopped and the particular configuration is selected for re-composition of the architecture (ii) *User- defined limit:* the user defines a maximum number for configurations to be executed upon adaptation. When the maximum number of configuration executions is reached, the

configuration with the best QoS is selected and (iii) *Most optimal QoS:* the QoS for all the possible configurations is calculated and the configuration with the best QoS is selected for re-composition of the architecture. The algorithm also enables the user to apply constraints on the configuration selection (line 12 and 13). The constraints filter out configurations with acceptable overall QoS but violating a specific user requirement (e.g., false positive rate should always be less than 0.05%). Irrespective of the stopping condition, Algorithm 5.1 always leads to a configuration that has as good a QoS as the configuration at which adaptation process is triggered. This is because Algorithm 5.1 follows a search-based approach where it searches the configuration space (consisting of different architectural configurations) with the aim to find a configuration that is either within the threshold or most optimal within the tested configurations. In the worst-case scenario, the algorithm will return the same configuration for which adaptation was triggered but will never return a configuration with a QoS worst than the configuration at which adaptation was triggered. However, as per our results presented in Section 5.5, we did not observe any such case where the algorithm did not return a configuration with a QoS better than the configuration at which adaptation was triggered.

Adaptation Properties: These properties measure the effectiveness of the adaptation approach [195]. Our approach is measured in terms of following properties – (i) *Adaptation Accuracy* measures how close the achieved QoS approximates to the ideal QoS upon adaptation. (ii) *Adaptation Stability* measures the ability of the adaptation approach to ensure that the managed system adapts under different scenarios instead of indefinitely searching for an optimal configuration (iii) *Adaptation Time* measures the speed of the adaptation approach i.e., how quickly the managed system adapts.

5.4 Implementation

After presenting our approach, we now provide a brief overview of the implementation approach as the details of the implementation are already reported in Chapter 3.

Instrumentation: We implemented *ADABTics* on Apache Hadoop [50] – a widely adopted big data processing framework. We used five ML algorithms in our BDCA system, which are implemented in a distributed fashion using Hadoop's machine learning library – Apache Mahout [52].

Execution modes: *ADABTics* has been executed in three different modes – ***S****tand****A****lone (SA),* ***P****seudo-****D****istributed (PD), and* ***F****ully-****D****istributed (FD).* This is because execution mode closely relates to response time, which is one of the key qualities considered in this study. *SA mode* mimics sequential execution, where Hadoop uses local file system and all jobs run in a single Java process. *PD mode* mimics distributed Hadoop cluster and stores data in HDFS but runs on a single machine. The machine used for SA and PD executions is an HP EliteBook 850 G5 configured with 2x Intel Xeon processors (2.60 GHz and 2.70 GHz), 8 GB RAM, 128 GB hard drive, and installed with CentOS 6.4 and JVM 1.7.0. *FD mode* distributes data processing among multiple machines. We configured a Hadoop cluster on OpenStack cloud for our experiments. The cluster consists of 17 nodes (16 slaves and 1 master) each running CentOS 6.4. Each slave node is equipped with 2GB RAM and 20GB disk space while the master node is equipped with 8GB RAM and 80GB disk space.

Managed System and Adaptation Controller: The various components of the managed system and adaptation controller have been implemented using Java MapReduce paradigm. Among several components presented in Section 5.3.1, we implemented only six components i.e., Removal of Duplicates (C1), Handling Missing Values (C2), Feature Selection (C3), Signature-based Detection (C4), Anomaly-based Detection (C5), and Alert Ranking (C6)) in the architectural configurations (Table 5.2) for evaluation of *ADABTics*. This is because a BDCA system can be designed using a multitude of

components as presented in Section 5.3.1, hence, implementing all components, and generating and executing subsequent architectural configurations is a challenging and time-consuming undertaking. Such an undertaking will only increase the number of configurations, which will have little (if none) impact on our evaluation results. We have implemented *Removal of Duplicates* using *NullWritable* class available in *org.apache.hadoop.io* library that detect and discard exactly similar records. For implementing *Handling Missing Values*, we used the mapper function to monitor each feature of each record and discard records with missing values. We implemented the *Feature Selection* component using InfoGain algorithm [142]. We followed the approach proposed in [158] for implementing *signature-based detection* component – leveraging C4.5 to generate signatures that can accurately detect known attacks. *Anomaly-based detection* is the core component and is instantiated in each configuration (Table 5.2). The component leverages ML algorithms, which are implemented using Mahout [52]. We separately used five state-of-the-art algorithms i.e., Kmeans (KM), XGBoost (XGB), Naïve Bayes (NB), Support Vector Machine (SVM), and Random Forest (RF) in the evaluation. For *Alert Ranking*, we employed the ranking model proposed in [157] that ranks the alerts based on the number of bytes embodied in each alert.

Datasets: We used four well-known security datasets (i.e., KDD [55], DARPA [56], CIDDS [57], and CICIDS2017 [58]) for the evaluation of our approach. The details of the datasets are available in Chapter 3.

5.5 Evaluation

In this section, we first outline the adaptation scenarios and configurations used for evaluation. We then formulate the evaluation objectives and finally present the results.

5.5.1 Adaptation scenarios

We have evaluated our approach under three different scenarios in which a BDCA system is likely to find itself. *BaseLine* – this is the (initial) state of the system in an operating environment where it is ensured that the system's achieved QoS is within the tolerance/acceptable level. Hence, there is no need of any adaptation. *Varying Data Quality* – the system is subjected to a change in the quality of security data (e.g., from KDD to DARPA). With the change in data quality, the achieved QoS is either forced out of the acceptable zone or remains in the acceptable zone. In the former case, adaptation is triggered to bring back the QoS within the acceptable level. *Varying Data Velocity* – the system is subjected to a change in the quantity of security data (e.g., from 500 MB to 1000 MB per unit time) to be processed at a given time. Similar to the scenario of varying data quality, the achieved QoS either gets out of the acceptable zone or remains within acceptable zone with the change in data velocity. The adaptation is triggered only in the case when achieved QoS gets out of the acceptable zone.

5.5.2 Configurations

As mentioned in Section 5.3.2, *ADABTics* requires the user to configure the values for ideal QoS, QoS threshold, constraint, and weights. The values configured for the sake of the evaluation are shown in Table 5.3. *Ideal QoS* is calculated by feeding the best possible values (i.e., target values) for each metric into Equation 5.1 generating value **0** for ideal QoS. For computing *QoS threshold*, we collected the values for each metric for the 20 cases (4 datasets × 5 algorithms) considered in our experiment. We then computed the mean for each metric shown in Table 5.3. The mean values are then fed to Equation 5.1, which generated a value of **1.07** for QoS threshold. For setting constraints, we first plotted boxplots for

Table 5. 3 Configuration values for QoS metrics

QoS Metric	Ideal	Mean	Constraint
Accuracy (%)	100	86.2	> 81.16
Detection Rate (%)	100	87.4	> 82.3
F1 Score (%)	100	86.9	> 81.4
False Positive Rate (%)	0	0.52	< 1.34
Prediction Time (sec)	0	17.3	< 28.4

each metric value obtained for the 20 cases to get the distribution of points. The upper bounds of the distribution are set as constraints for the positive QoS metrices (e.g., prediction time < 17sec) while the lower bounds of the distribution are set as constraints for the negative QoS metrices (e.g., Accuracy > 78.16). We assigned same weight (i.e., 1) to all metrices.

5.5.3 Evaluation objectives

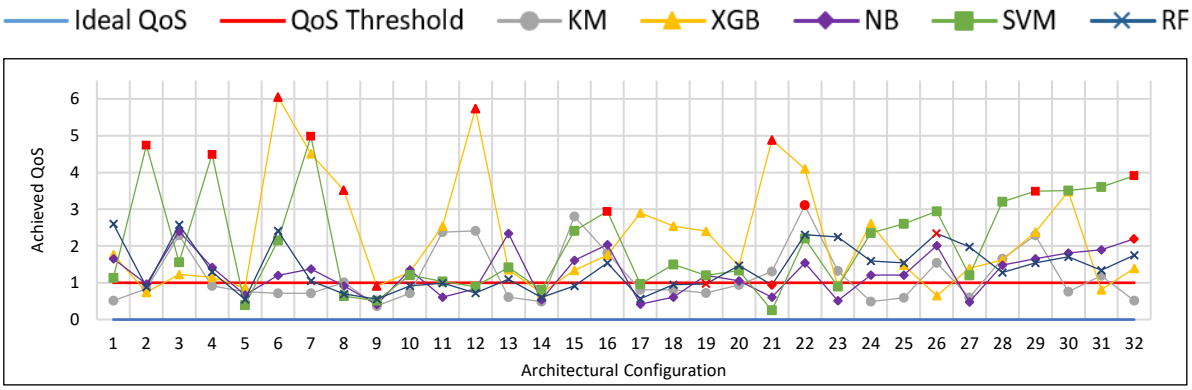
The evaluation presented in this section aims to answer the following Research Questions (RQ).

- ***RQ1:*** *How close to the ideal QoS does the QoS of BDCA system reach using ADABTics (i.e., adaptation accuracy)?*
- ***RQ2:*** *How often is ADABTics successful in making BDCA system adapt to the changes (i.e., adaptation stability)?*
- ***RQ3:*** *How long does it take for ADABTics to make a BDCA system adapt to the changes (i.e., adaptation time)?*

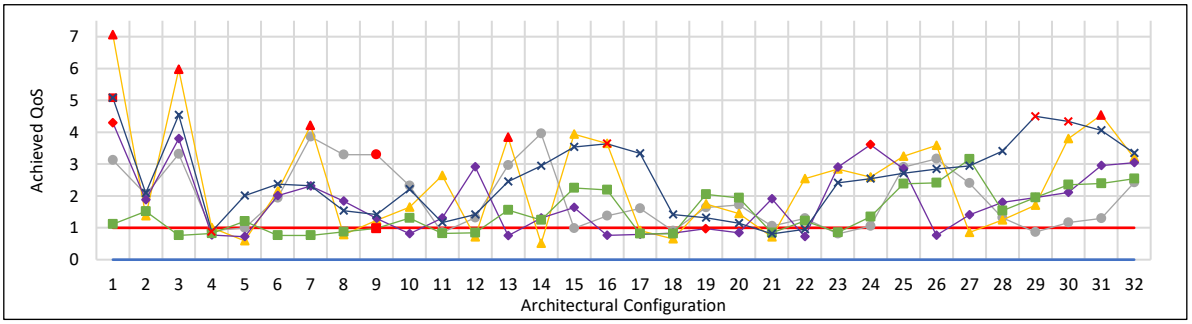
5.5.4 Results

5.5.4.1 RQ1: Adaptation accuracy

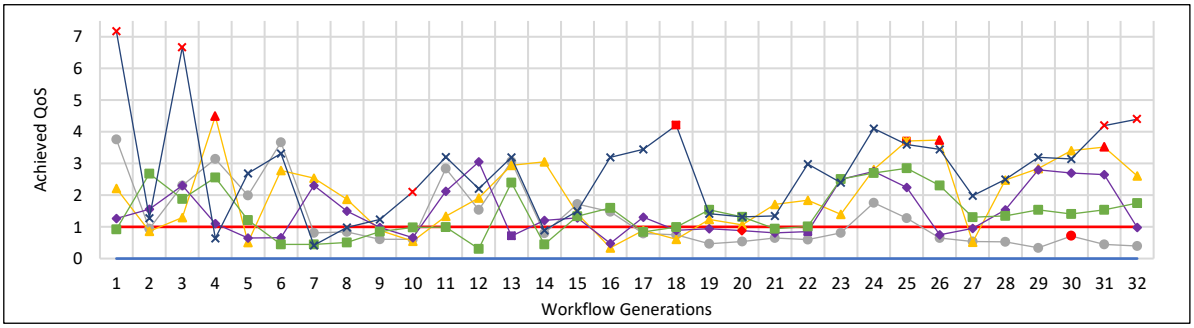
Adaptation accuracy is measured by how close is the best QoS (among several QoS values for a specific case) to the ideal QoS. Figure 5.4 shows the QoS achieved by the BDCA system for each of the dataset and for each of the ML algorithm in various architectural configurations. The x-axis in Figure 5.4 specifies the ID of the architectural configuration from Table 5.2 and the y-axis denotes the QoS achieved by the system at runtime as calculated using Equation 5.1. For the 20 experimental cases (i.e., 4 datasets × 5 algorithms), the average adaptation accuracy achieved by *ADABTics* is 0.42. The best accuracy (i.e., 0.07) is achieved with CICIDS2017 + XGB in configuration 14 (Figure 5.4d). The metrics values for the best adaptation accuracy are detection rate 95, F1 score 94.8, accuracy 94.61, FPR 0.11, and prediction time 25.94 sec. The components operationalized in configuration 14 are C3 (Feature Selection), C4 (Signature-based Detection), and C5 (Anomaly-based Detection). Among datasets, the most accurate (average) results are obtained for CICIDS2017 (0.13) followed by CIDDS (0.37), KDD (0.44), and DARPA (0.72). This trend is in line with the number of features in each dataset. The system achieves high detection rate, F1 score, and accuracy for CICIDS2017, which has the largest number of features (i.e., 82). Hence, these metrices dominate Equation 5.1 to generate lower QoS values for CICIDS2017. Similar to datasets, the adaptation accuracy varies among the ML algorithms employed in the system with SVM taking the lead to achieve an adaptation accuracy of 0.37 on average. The remaining algorithms (i.e., XGB, KM, NB, and RF) achieves an adaptation accuracy of 0.39, 0.42, 0.43, and 0.47 respectively. The configurations violating the applied constraints are specified by the red dots in Figure 5.4. As can be seen, the configurations generating the most accurate QoS do not violate any constraint. Therefore, adaptation accuracy is not affected by the applied constraints. Figure 5.5a illustrates the QoS achieved for each dataset under different execution modes. FD mode with a mean



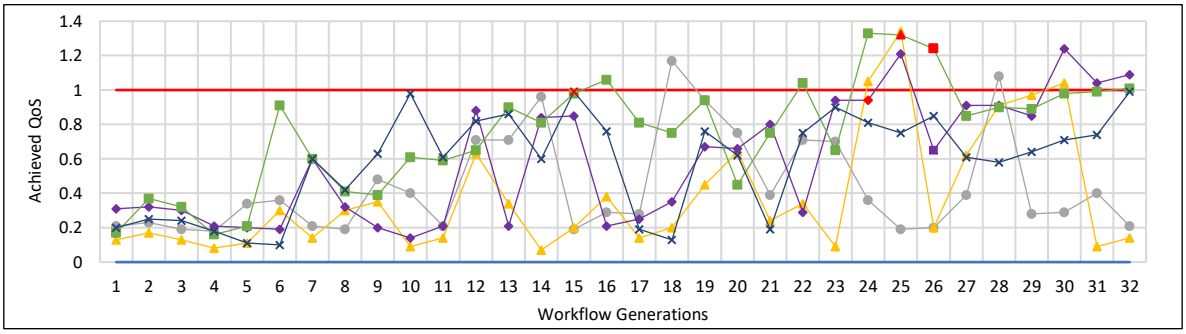
(a) Achieved QoS values with KDD dataset



(b) Achieved QoS values with DARPA dataset



(c) Achieved QoS values with CIDDS dataset



(d) Achieved QoS values with CICIDS dataset

Figure 5. 4 Achieved QoS for the 32 configurations with four datasets and five ML algorithms. Achieved QoS is calculated using Equation 5.1.

adaptation accuracy of 0.45 outclasses SA and PD mode, which achieves mean adaptation accuracy of 0.48 and 0.49 respectively. However, this difference is not in line with the computational power (16 nodes to FD and 1 node to SA and PD) allocated to each mode. This is because the fully distributed implementation of Hadoop is more efficient for larger datasets [50, 196, 197]. In our implementation, this is evident from the difference observed for the large size dataset as compared to the small size

dataset. For example, the average adaptation accuracy for the three modes in case of CIDDS (which is a large size dataset) is 0.32 (FD), 0.36 (SA), and 0.37 (PD) as compared to 0.5 (FD), 0.52 (SA), and 0.53 (PD) for KDD (a small size dataset).

5.5.4.2 RQ2: Adaptation stability

Adaptation stability is measured by the number of times achieved QoS falls within the threshold region. The greater the adaptation stability, the greater are the chances that the system will adapt upon activation of adaptation trigger. It is important to note that adaptation stability depends upon the choice of threshold range [195]. The results presented here are in accordance with our threshold range discussed in Section 5.5.2. For the 640 QoS points illustrated in Figure 5.4, 295 falls within the threshold region. This way *ADABTics* achieves an adaptation stability of around 47.3% (i.e., 303/640). The result is important from two perspectives. First, it reveals that 52.7% of BDCA's configuration are not able to deliver the desired QoS, which motivates the need for adaptation. Second, the result illustrates that there are 47.3% chances of making the system adapt to the operating environment. The highest stability is achieved with CICIDS2017, where 145 out of 160 (32 configurations × 5 algorithms) QoS points lies within the threshold region. This is followed by KDD, CIDDS, and DAPRA for which 59, 56, and 43 QoS points lies with the threshold region respectively. Among classifiers, KM achieves the best adaptation stability (i.e., 58.3%) on average, which is followed by NB, SVM, XGB, and RF with a mean stability of 49.1, 46.6, 41.6, and 41.2 percent respectively. The QoS values for configurations that violate any of the applied constraints lies primarily out of the threshold region. Figure 5.4 shows that only 11 out of 303 QoS values lying within the threshold region violates a constraint. This brings down the adaptation stability of *ADABTics* to 45.6% for the scenario where constraints are considered. The adaptation stability is highest (47.3%) in FD mode and lowest (41.8%) in PD mode.

5.5.4.3 RQ3: Adaptation time

Adaptation time underlines the speed with which our adaptation approach makes the BDCA system adapt to the changes. As the reader may recall, the stopping conditions in our algorithm (i.e., Algorithm

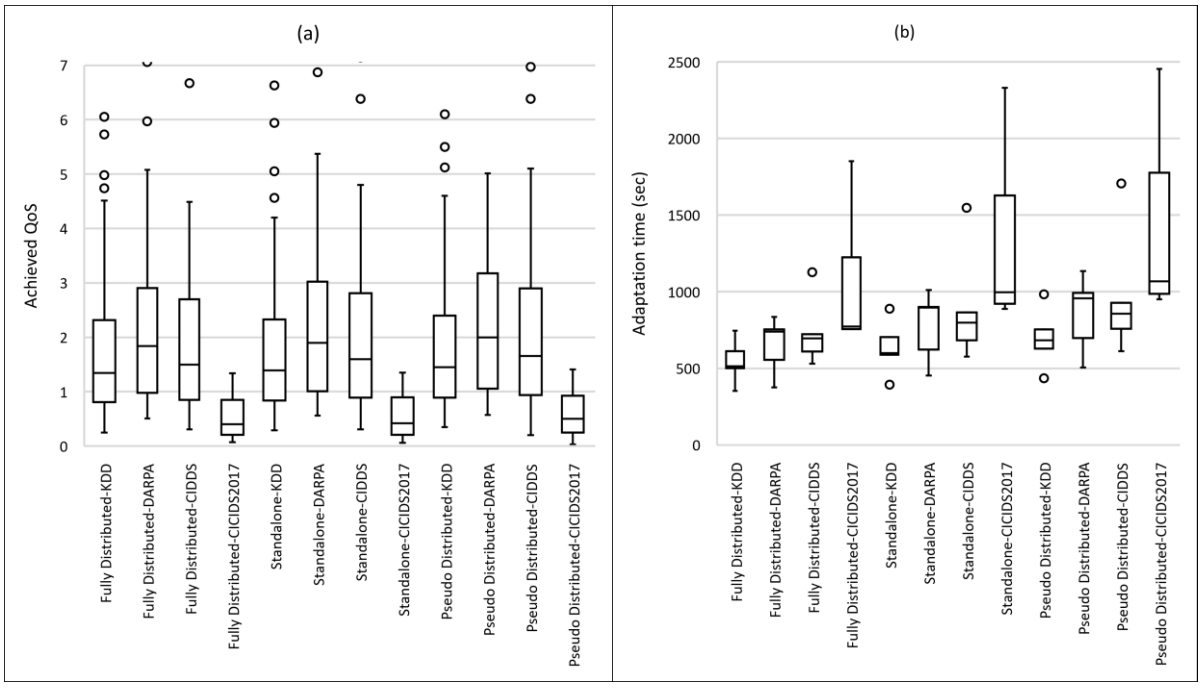


Figure 5. 5 Achieved QoS and adaptation time in Fully Distributed, Standalone, and Pseudo Distributed mode

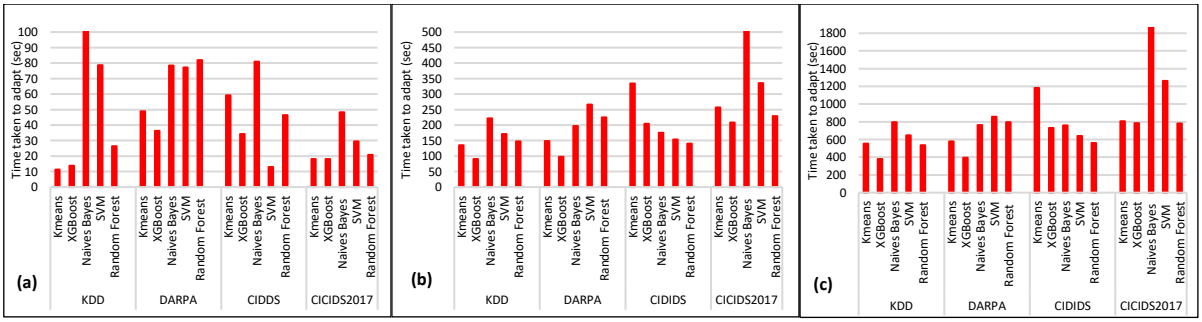


Figure 5. 6 Adaptation time of *ADABTics* with respect to the stopping condition i.e., (a) QoS satisfaction (b) User-defined limit and (c) Most optimal QoS

5.1) put a barrier on the execution time of the search process for selecting the optimal configuration. Therefore, we report our results with regards to the three stopping conditions – (i) *QoS satisfaction* (ii) *User-defined limit* (iii) *Most optimal QoS*. The time taken by *ADABTics* to adapt the system under the three operating conditions is shown in Figure 5.6. The average adaptation time with *QoS satisfaction* condition is 4.6 times lower than *User-defined limit* condition and 17.3 times lower than *Most optimal QoS* condition. This trend is in line with the number of configurations to be executed under each stopping condition. It is important to note that unlike *User-defined limit* (set as 10 in this experiment) and *Most optimal QoS*, there is no control over the number of configuration execution under *QoS satisfaction* condition. This is because under *QoS satisfaction* condition, as soon as the algorithm finds a configuration with a QoS within the threshold, the algorithm selects the configuration and stops the search process. It is found that under *QoS satisfaction* condition, the adaptation time is lowest with DARPA (26.9 sec), which is followed by KDD (46.2 sec), CIDDS (46.7 sec), and CICIDS2017 (64.5 sec). With *User-defined limit* and *Most optimal QoS* conditions, the system adapts fastest with KDD followed by DARPA, CIDDS, and CICIDS2017. This trend shows that under these two conditions, the adaptation time increases proportionally with the increase in the number of records in the dataset (presented in Table 3.3). The average adaptation time taken by various ML techniques under *QoS satisfaction* condition is XGB (25.5 sec), KM (34.3 sec), RF (43.9 sec), SVM (49.5 sec), and NB (77.17 sec). The enforcement of constraints does not have any significant impact on adaptation time under second and third stopping condition as the constraints are checked after the QoS is calculated for the configuration (Algorithm 5.1). Although the constraints do not impact the adaptation time even under *QoS satisfaction* condition in our experiments, this finding cannot be generalized. This is because it is quite likely that a different set of constraints might be violated by the first configuration with satisfied QoS, in which case the adaptation time will be negatively impacted. Figure 5.5b shows the impact of execution mode on the adaptation time for *Most optimal QoS* condition. Unlike the marginal impact of execution mode on adaptation accuracy (where accuracy related metrices dominate Equation 5.1), the impact of execution mode is quite significant for adaptation time. The average adaptation time with FD mode (751.7 sec) is 17.8% faster than SD mode (914.5 sec) and 24.3% faster than PD mode (993.4 sec).

5.6 Discussion

In this section, we discuss the results with respect to the optimization achieved in accuracy and prediction time and the trade-off between adaptation accuracy and adaptation time.

5.6.1 Quantifying the achieved optimization

We now turn to the question (related to our goal) that how much *ADABTics* optimizes the accuracy and prediction time. We try to answer this question by comparing the accuracy and prediction time of

Table 5. 4 Accuracy and prediction time of our BDCA system before and after adaptation

Stopping Condition	Experimental Case	Before Adaptation		After Adaptation		Adaptation Time (sec)
		Accuracy (%)	Prediction Time (sec)	Accuracy (%)	Prediction Time (sec)	
QoS Satisfaction	KDD	86.2	21	93.4	13.28	46.2
	DARPA	78.06	14.66	85	14.68	26.9
	CIDDS	79.4	23.9	82.9	16.8	46.7
	CICIDS2017	90.45	32.5	94.5	26.91	64.5
User-defined Limit	KDD	86.2	21	95.5	13	152.8
	DARPA	78.06	14.66	86.8	14.24	186.8
	CIDDS	79.4	23.9	83.16	17.5	201.24
	CICIDS2017	90.45	32.5	94.6	26.2	307
Most Optimal QoS	KDD	86.2	21	94.52	13.3	545.3
	DARPA	78.06	14.66	86.8	14.18	651.8
	CIDDS	79.4	23.9	83.22	14.6	736
	CICIDS2017	90.45	32.5	94.6	25.94	1073

the BDCA system exactly before and after the adaptation. Table 5.4 shows the mean values for accuracy, prediction time, and adaptation time for the 20 experimental cases under the three different stopping conditions before and after the adaptation. Under *QoS satisfaction* condition, adaptation improves accuracy and prediction time by 5.43% and 22.1% respectively. The average improvement recorded under *User-define limit* and *Most Optimal QoS* conditions are 6.49% and 6.26% in accuracy and 23% and 26.11% in prediction time respectively. To investigate whether the achieved improvement is statistically significant, we applied Wilcoxon signed ranked test [198] (with significance level $\alpha = 0.05$) on the two set of values (i.e., before and after adaptation). The test generated p values of 0.00222, 0.0021, 0.0019 for accuracy and 0.0096, 0.0097, and 0.00998 for prediction time under *QoS satisfaction*, *User-defined*, and *Most Optimal QoS* condition respectively. All six values are lower than α (i.e., $p<0.05$), which concludes that the improvement for both accuracy and prediction time is statistically significant under all three stopping conditions.

5.6.1 Trade-off between adaptation accuracy and adaptation time

Most of the prior work on architecture-driven adaptation (e.g., [199] ,[200]) takes a short sighted view to adapt for the the most optimal QoS. Such approaches though are successful in achieving optimal QoS, but at the cost of increased adaptation time. Unlike prior work, our approach takes a long-sighted view and employs the flexibility to do a trade-off between adaptation accuracy and adaptation time. As evident from Table 5.4, the accuracy and prediction time are the most optimal under *Most Optimal QoS* condition followed by *User-defined limit* and *QoS satisfaction* conditions. However, the adaptation time is lowest under *QoS satisfaction* condition followed by *User-defined limit* and *Most Optimal QoS* condition. Therefore, a long-sighted view is required to give the opportunity to the user to employ the adaptation intelligently based on his know-how of the operating environment. Accordingly, our approach enables the user to leverage his basic knowledge of the operating environment. If the environment is experiencing frequent changes, adaptation is triggered very often. Therefore, the user opts for *QoS satisfaction* condition to minimize adaptation time. For an average frequency of changes, the user chooses the *User-defined limit* to customize the adaptation time. Finally, if the environment rarely changes, the user chooses *Most Optimal QoS* condition to achieve highest level of optimization.

5.7 Threats to Validity

Internal Validity: The initial configuration of *ADABTics* (e.g., setting up the thresholds and constraints) relies on the expertise of security experts. If setup unrealistically and not aligned with the underlying operating environment, *ADABTics* may degrade accuracy and response time instead of optimizing them. However, the assumption is common to most of the research on adaptation approaches (e.g., [199, 201, 202]). We set the value of threshold in Section 5.5.2 based on the findings from the 20 use cases (i.e., 4 datasets × 5 ML algorithms), however, such a value can also be selected based on expert opinion, which will further improve the validity of the our findings. ***Construct Validity:*** The configuration selection algorithm (Algorithm 5.1) has to explore all possible configurations before finding the configuration with the most optimal QoS, which makes it an NP-hard problem [202]. Whilst most architecture-based optimization algorithms face this limitation, some techniques (e.g., market-based control [202]) have been recently proposed to mitigate the problem. We leave the incorporation of such techniques into *ADABTics* as a future work. Furthermore, the configurations can also be defined automatically using a heuristic-based approach instead of pre-defining them as reported in this chapter. ***External Validity:*** Given that a BDCA system can be architected using different set of components and different big data frameworks (e.g., Spark and Storm), we cannot guarantee the generalization of our findings. However, it is worth noting that the goal of this chapter is not to generalize the results, but rather to demonstrate the ability of *ADABTics* to ensure optimal accuracy and response time. Adaptation scenarios, for which *ADABTics* is evaluated, are not limited to what is shown in Section 5.5.1. We assert that evaluating *ADABTics* for additional adaptation scenarios (e.g., slow and steady change in data quality or data velocity) will add to the generalization of our findings.

5.8 Related Work

This section positions the work reported in this chapter with respect to related studies.

As discussed in Chapter 2, BDCA has been an active area of research and practice. A large number of Hadoop-based BDCA systems (e.g., [40, 170, 203-206]) have been proposed [1]. Of these, some (i.e., [203-205]) are single machine system while others (i.e., [40, 170, 206])) support distributed processing. The incorporation of ML algorithms is limited to two to three algorithms in these systems. Except [205], which incorporates three ML algorithms (i.e., NB, SVM, and RF), the rest of the systems incorporate either one ([204, 206]) or two ([40, 170, 203]) algorithms. Furthermore, only one system ([170]) is evaluated with two security datasets (i.e., KDD and CMDC2012) while the rest with only one dataset (i.e., KDD). In contrast, our implemented BDCA system incorporates several ML algorithms (i.e., KM, XGB, NB, SVM, and RF) and evaluated with four security datasets (i.e., KDD, DARPA, CIDDS, and CIDIDS2017) in three execution modes.

Research efforts have been invested in proposing and evaluating BDCA architectures ([188, 196, 197, 207, 208]). However, the existing architectures are of a higher abstraction level and do not follow the guidelines of component-based architectural style [190]. Ours is the first effort to propose a component-based architecture for BDCA, which improves system's structure and flexibility by allowing addition and removal of components at runtime.

The optimization of quality (e.g., accuracy and response time) is a well-trodden phenomenon explored in different domains such as embedded systems [209], cloud-based systems [210], radar systems [211], and object detection systems [212]. Whilst [212] investigates various architectural options (e.g., feature extractor and image size) where accuracy can be traded for response time, [209-211] present and

evaluate approaches (i.e., controlling length of cryptographic keys, distributed simulations and symbiotic feedback loops, and service lever configurations based on tasks respectively) for optimizing the QoS. Unlike prior works, our work reported in this chapter is in a different domain (i.e., BDCA) and explores a different approach (i.e., architecture-driven adaptation with stopping conditions) that aims to ensure optimal accuracy and response time.

Unlike control-based adaptation, there is little understanding and exploration of architecture-driven adaptation in different domains of software-intensive systems [193]. However, some recent efforts underline the increased interest in the application of architecture-driven adaptation in real-world software systems. This is evident from exploring its application in different domains (i.e., robotic systems [199], network monitoring [193], IoT [200], and business transactions [213]). Motivated by such efforts and the utmost need, this work presented in this chapter is the first of its kind that presents, applies, and evaluates architecture-driven adaptation in a different but an important domain (i.e., BDCA).

5.9 Chapter Summary

The frequent changes in the operating environment of a BDCA system impacts system's accuracy and response time. In this chapter, we first expose the impact of such environmental changes. We then propose *ADABTics* – an architecture-driven adaptation approach that recomposes the architecture of the BDCA system at runtime in accordance with the environmental changes to ensure optimal accuracy and response time. We evaluate our approach through a Hadoop-based BDCA system using four security datasets and five machine learning algorithms. Our evaluation demonstrates the effectiveness of our approach in terms of adaptation accuracy, adaptation stability, and adaptation time. The adaptation approach optimizes the accuracy and response time of the BDCA system by 6.06% and 23.7% respectively.

Chapter 6

Scalable Adaptation of Big Data Cyber Security Analytics Systems

Related publication:

This chapter is based on our paper published in *International Conference on Engineering of Complex Computer Systems (ICECCS 2019)* "QuickAdapt: Scalable Adaptation for Big Data Cyber Security Analytics"[4].

Chapter 5 uncovers that the choice of architectural configuration is directly related to the type of security dataset. Therefore, it is important to adapt a BDCA's system architectural configuration in accordance with the type of the security dataset. However, it is also crucial to consider the large search space of architectural configurations. Searching a large space of configurations for potential adaptation incurs an overwhelming adaptation time, which may cancel the benefits of adaptation. For instance, *ADABTics* (presented in the previous chapter) takes around 751 sec on average to select the optimal configuration for a BDCA system. In this chapter, we build on the work presented in Chapter 5 to present a scalable adaptation approach (i.e., *QuickAdapt*) that enables quick adaptation of a BDCA system. *QuickAdapt* uses descriptive statistics (e.g., mean and variance) of security events data and fuzzy rules to quickly (re)compose a system with a set of architectural components to ensure optimal accuracy and response time. We have evaluated *QuickAdapt* for a distributed BDCA system using four security datasets. Our evaluation demonstrates that the adaptation time for *QuickAdapt* is significantly less than the adaptation time for *ADABTics*.

6.1 Introduction

In Chapter 5, we presented an architecture-driven adaptation approach, *ADABTics* [5], that selects architectural configuration at runtime to enable a BDCA system to ensure optimal accuracy and response time. In *ADABTics*, the search process for finding the optimal architectural configuration operates with respect to three stopping conditions i.e., *QoS Satisfaction*, *User-defined limit*, and *Most Optimal QoS*. Whilst the adaptation under QoS Satisfaction condition is quick, it compromises on the adaptation accuracy (i.e., selecting the optimal configuration). On the other hand, adaptation accuracy is quite high under Most Optimal QoS condition, but it takes a longer time (i.e., 751.5 sec on average) to adapt the system as compared to adaptation time under QoS Satisfaction condition. Finally, with User-defined limit, adaptation time and adaptation accuracy depend upon the number/limit of search iterations set by the user. The higher the limit, the higher is the accuracy and adaptation time. It is important to note that adaptation time is not same as response time and adaptation accuracy is not the same as (classification) accuracy. Whilst response time measures how quickly the BDCA system classifies the security event data into benign and malicious classes, adaptation time measures the time

our adaptation approach requires to adapt the system as per the changes in the operating environment. Similarly, (classification) accuracy measures how accurately our BDCA system classifies the security event data into benign and malicious classes. On the other hand, adaptation accuracy measures how accurately our adaptation approach selects the optimal architectural configuration with which our BDCA systems archives optimal (classification) accuracy and response time. The adaptation time under Most Optimal QoS condition is high for two reasons: (a) the search space, from which *ADABTics* selects the optimal configuration, is large (i.e., 32 configurations considered in Chapter 5) due to several available components and their subsequent combinations and (b) *ADABTics* sequentially executes each configuration within the search space at runtime to find its effectiveness and ultimately selecting the most optimal configuration. This situation has stimulated a question to be addressed: "***How to enable a BDCA system to quickly adapt to the changes in the security events data?***"

In this chapter, we present *QuickAdapt*, a runtime adaptation approach that optimizes accuracy and response time through a quick adaptation of a BDCA system with respect to changes in security events. Similar to *ADABTics* [5], *QuickAdapt* exploits the component-based architecture of a BDCA system to add or drop components at runtime. Whilst *ADABTics* and other related adaptation approaches (e.g., [5, 214-216]) face the issue of high adaptation time due to searching a large configuration space, *QuickAdapt* takes a direct approach of monitoring and extracting the necessary insights from security events to realize adaptation. *QuickAdapt* uses Kolmogorov-Smirnov test [217] to monitor security events data and trigger adaptation upon a significant drift in the data. Once adaptation is triggered, *QuickAdapt* leverages a set of fuzzy rules, developed using the Wang-Mendel's fuzzy rule generation method [218], to find the most optimal configuration from the search space. In principle, the rules express and map the statistics with linguistic rules and membership functions, hence, decide whether a component should be operationalized keeping in view the type of input data. Thus, the rules draw a relationship between the input data and a system's architecture. For example, duplicates removal component should not be operationalized for processing data with very small percentage of duplicate instances. The components selected based on the rules eventually form the configuration to be adopted. We have evaluated *QuickAdapt* using a Hadoop-based BDCA system deployed on a large-scale cluster. Like Chapter 5, we use four security datasets in the evaluation. Our evaluation shows that in comparison to *ADABTics* [5], *QuickAdapt* reduces the adaptation time by 105 times with minimal impact on adaptation accuracy.

Contributions: In this chapter, we make the following novel contributions.

1. Present *QuickAdapt* - a runtime adaptation approach that enables quick adaptation of a BDCA system
2. Formulate fuzzy rules that are used by *QuickAdapt* for the runtime selection of components
3. Implement and evaluate the effectiveness of *QuickAdapt* through rigorous experimentation

Chapter Organization: Section 6.2 presents *QuickAdapt*. Section 6.3 briefly describes the implementation details of *QuickAdapt*. Section 6.4 reports the evaluation setup and evaluation results. Section 6.5 discusses the results. Section 6.6 outlines the threats to the validity of the work presented in this chapter. Section 6.7 positions the novelty of *QuickAdapt* with respect to related work. Section 6.8 concludes the chapter.

6.2 QuickAdapt Overview

QuickAdapt consists of two parts – BDCA component-based architecture and scalable adaptation. The BDCA component-based architecture is already presented in Chapter 5. *QuickAdapt* exploits the same component-based architecture to materialize scalable adaptation. The various facets of the adaptation approach of *QuickAdapt* are presented in the following.

6.2.1 Adaptation goal

QuickAdapt aims to "*enable a BDCA system to quickly adapt to the changes in the input data*". Similar to Chapter 5, we use Quality-of-Service (QoS) [219] as a measure to quantify the accumulated impact on accuracy and prediction time. We measure accuracy through four well-known metrics i.e., F1 score (F1), False Positive Rate (FPR), Detection Rate (DR), and Accuracy (Acc) and Prediction Time (PrT) as the time taken by the system to detect attacks in the testing dataset [5].The details of the metrics are described in Table 4.1.

6.2.2 Reference inputs and measured outputs

Reference inputs specify the target that our *QuickAdapt* aims to achieve. Similar to most of the research on QoS-aware self-adaptation (e.g., [214, 215]), our reference input is the *target QoS* that our approach attempts to achieve. As mentioned in Chapter 5, we measure QoS using the utility function (Equation 5.1), which weighs and sums the values for the considered quality metrics (e.g., detection rate) presented in Table 4.1. The utility function aims to maximize desirable quality attributes (i.e., accuracy, detection rate, and F1 score) and minimize undesirable quality attributes (i.e., false positive rate and prediction time). We use Equation 5.2 and Equation 5.3 to scale undesirable and desirable quality attributes respectively. Whilst the reference inputs are configured by a user, the measured outputs are the values monitored as the system outputs. The measured outputs are then compared with the reference inputs to evaluate whether the system has achieved the target. In our approach, the measured output is the *achieved QoS* that is the QoS system delivers after it has adapted to the change.

6.2.3 Adaptation trigger

The adaptation trigger specifies the condition upon which the adaptation process is started. We monitor the input data as the changes in input data significantly impact the accuracy and prediction time as demonstrated in Chapter 5. For quantifying the change in input data, we use Kolmogorov-Smirnov (KS) test [217] that determines the divergence between two data distributions (i.e., D_{old} and D_{new}) shown in line 1 of Algorithm 6.1 based on the mean values for each feature. The test, undertaken by the drift calculator as shown in Figure 6.4, detects change both in location and shape of the two data distributions, hence, makes the detection of data change more generic. After calculating the divergence, we apply *p-value* test [220] with a significance level 0.05 to determine whether the divergence is statistically significant. In case of a significant divergence, adaptation is triggered.

6.2.4 Control actions

Upon triggering an adaptation, the control actions are initialized to enable a system to achieve the target state. Given our component-based architecture, *QuickAdapt* relies on architecture reconfiguration for adapting a system. The architecture configuration is specified by the components operationalized as presented in Table 5.1. Equation 6.1 is used to calculate the total number of possible

Algorithm. 6.1. *QuickAdapt* - Adaptation Algorithm

Input: $C = \{C_1, C_2, C_3, \ldots, C_N\}$ // *Set of components*
D_{old} = Old data distribution
D_{new} = New data distribution
$R = \{R_1, R_2, R_3, \ldots, R_N\}$ // *Set of fuzzy rules*

Output: AC_{opt} = Optimal Architectural Configuration

Steps:
1. Calculate divergence D (D_{old}, D_{new})
2. Calculate *p-value* for divergence D
3. **If** *p-value* <0.05 **then**
4. $AC_{opt} \leftarrow \emptyset$ // *Initialize to empty set*
5. Calculate V_1 to V_{13} for D_{new} //*Input Variable V*
6. Normalize V_1 to V_{13}
7. **for** i = 0 to i = N **do**
8. Execute R_i
9. **If** R_i == SELECT **then**
10. AC_{opt}.Append (C_i) // *Add component*
11. **end**
12. **end**
13. **end**

configurations in the search space. Equation 6.1 uses mathematical combinations to calculate the total number of configurations, where C_N denotes the total number of configurations, a specifies the number of components before and b specifies the number of components after the Anomaly-based Detection component, g and h specifies the combination of elements from the total set. For the set of components shown in Figure 5.3, a=11 and b=3, which generates a search space of C_N = 16,376. The reconfiguration process aims to select components (for forming a configuration) that are most effective for the input data.

$$C_N = \sum_{g=1}^{a} \binom{a}{g} + \sum_{g=1}^{a} \binom{a}{g} \times \sum_{h=1}^{b} \binom{b}{h} \tag{6.1}$$

Component Selection: The selection of components for creating an effective architectural configuration is a challenging task. This is because the effectiveness of a component depends upon several factors such as the size of input data, the number of classes in input data, the number of features in each instance, and the number of duplicate instances [189, 221]. Given several factors that are considered in component selection, it is challenging to make a binary decision about component selection. Hence, we cannot be 100% sure about the selection of a component. Instead of binary logic, we therefore, leverage fuzzy logic in our decision-making process about component selection. Fuzzy logic is a powerful tool for decision making that involves some degree of subjectivity and impreciseness [215]. Fuzzy logic considers all underlying factors in the process of fuzzification and defuzzification to generate a decision with a specific degree of certainty. For example, a component should be selected with a 75% confidence. Fuzzy logic maps *input variables* to *output variables* through a *membership function*.

Input variable: We consider 13 *input variables* that are extracted through descriptive analytics [222] of the input data (line 5 of Algorithm 6.1). The descriptive analytics analyse data to extract information such as mean, variance, and number of features etc. The input variables are described in Table 6.1. The input variables are evaluated as LOW, MEDIUM-LOW (MED-LOW), MEDIUM, MEDIUM-HIGH (MED-HIGH), and HIGH. The values for all input variables are scaled in the range [0,1]. ***Output variable:*** The *output variable* is the decision, i.e., whether or not a component should be selected. The decision is denoted as SELECT or NOT SELECT. The membership function is applied on *input variables* to determine the degree of membership for *output variable*. We used triangular membership function for all the rules with 50% membership for each subset e.g., LOW and MED.

Table 6. 1 Input variables and their descriptions

S#	Input Variable	Description
1	% of duplicates instances	Percentage of duplicates instances i.e., instances having exactly same values for all features
2	% of missing values	Percentage of instances with values missing for at least one feature
3	% of incorrect values	Percentage of instances with incorrect value (e.g. number of source bytes = -20) for at least one feature
4	% of valueless data	Percentage of instances having no relevance for security analytics e.g., instances associated with zero-bytes used for handshaking
5	Test sample size	Number of instances in the testing dataset
6	Train sample size	Number of instances in the training dataset
7	% of identifier and timing information	% of instances having unique identifier and time window e.g., *ID#1 22:30 instance_1*, *ID#2 22:31*
8	Number of features	Number of features in each instance of the data
9	Feature variance	Variance among the features of the data
10	Feature correlation	Correlation among the features of the data
11	% of categorical features	Percentage of categorical features in the data
12	Source bytes variance	Variance among the values of the source byte feature in the data
13	Destination bytes variance	Variance among the values of the destination bytes features in the data

Fuzzy rules: Finally, we formulate fuzzy rules for component selection that describe the relationship between observation (input variable) and conclusion (output variable). Table 6.2 presents the fuzzy rules along with a brief rationale for each rule. Each rule is defined in the form of a logical implication $p \rightarrow q$, where p is the observation and q is the conclusion. The symbols '$\wedge$' and '$\vee$' denote AND and OR operator respectively. The conclusion 'SELECT' as part of the rule implies that if the associated conditions hold true, the component is selected, otherwise, the component is not selected in the architectural configuration. Since anomaly-based detection is the component selected in each configuration as described in Chapter 5, Table 6.2 does not have a fuzzy rule for the selection of anomaly-based detection component. We used the well-known Wang-Mendel's method [218] for generation of the fuzzy rules. The main advantage of Wang-Mendel's method is that it is fast and combines the importance of qualitative information and numerical (quantitative) information for generating the rules. For qualitative information, we used the insight gathered through our literature review reported in Chapter 2. For numerical information, we leveraged the information generated through our large-scale experimentation on evaluating the impact of the BDCA components with respect to three contextual factors reported in Chapter 3. We briefly outline the steps followed to generate the rules. *Step 1:* divide the input and output spaces of the numerical data into fuzzy subsets (Figure 6.1) *Step 2:* generate fuzzy rules from the data *Step 3:* Assign a degree of importance to each rule to resolve conflicts among the rules *Step 4:* refine the rules based on qualitative insight gathered from the literature review reported in Chapter 2 *Step 5:* map the input variables to the output variables using the refined rules. While designing the rules, it is also considered that the components depending on each other operate at the same time and components contradicting each other (e.g., feature generation and feature transformation) do not operate at the same time.

Fuzzy rule execution: We illustrate the execution of fuzzy rules (line 7-12 of Algorithm 6.1) by *QuickAdapt* with the help of a decision about the selection of a component i.e., signature-based detection for CIDDS dataset. The fuzzy rule for the selection of signature-based detection consists of two input variables i.e., *test sample size* and *number of features*. The scaled values of *test sample size* and *number of features* for CIDDS is 0.37 and 0.45 respectively deducted through descriptive statistics of the dataset and shown in Figure 6.2. The rule matrix generated from the rule that leads to the decision of SELECT (S) or NOT SELECT (NS) is shown in Figure 6.1. The degree of membership of *test sample size* is 1 for MED- LOW and 0 for other subsets (i.e., LOW, MED, MED-HIGH, HIGH) as shown in Figure

Table 6. 2 Fuzzy rules for the selection of components

Component	Fuzzy Rule	Rationale
Removal of Duplicates	If (% *of duplicate instances* is MED ∨ % *of duplicate instances* is MED-HIGH ∨ % *of duplicate instances* is HIGH) ∧ (test *sample size* is MED ∨ *test sample size* is MED-HIGH ∨ *test sample size* is HIGH) → SELECT	If the data does not have large number of duplicate instances and the data size is large, there is no need for passing the data through the removal of duplicates component. This is because such processing will only lead to increase in response time without any gain.
Handling Missing Values	If (% *of instances with missing values* is MED ∨ % *of instances with missing values* is MED-HIGH ∨ % *of instances with missing values* is HIGH) ∧ (*test sample size* is MED ∨ *test sample size* is MED-HIGH ∨ *test sample size* is HIGH) → SELECT	If the size of testing data is large with only few instances with values missing for a feature, passing such data through handling missing values component only leads to extra processing without any gain in accuracy.
Removal of Incorrect Data	If (% *of instances with incorrect values* is MED ∨ % *of instances with incorrect values* is MED-HIGH ∨ % *of instances with incorrect values* is HIGH) ∧ (*test sample size* is MED ∨ *test sample size* is MED-HIGH ∨ *test sample size* is HIGH) → SELECT	If only a small percentage of data has incorrect values and the size of test data is large, the time spent in processing the data for removal of instances with incorrect values surpass the gain expected in accuracy.
Removal of Valueless Data	If (% *of instances with valueless data* is LOW-MED ∨ % *of instances with valueless data* is MED ∨ % *of instances with valueless data* is HIGH) ∧ (*test sample size* is MED ∨ *test sample size* is MED-HIGH ∨ *test sample size* is HIGH) → SELECT	If test data has large amount of valueless data that does not contribute to attack detection, the valueless data should be removed otherwise processing the whole data for removing valueless data only increases the response time
Data CutOff	If (*sample size* is HIGH) ∧ (quantity of *identifier and timing information* is MED ∨ quantity of *identifier and timing information* is MED-HIGH ∨ quantity of *identifier and timing information* is HIGH) → SELECT	Data from same process or connection should be reduced to improve response time only if the size of test data is large and has identifier information is available as removal of such instances does pose a risk of attack detection evasion
Feature Selection	If (# *of features* is MEDIUM ∨ # *of features* is MED-HIGH ∨ # *of features* is HIGH) ∧ (*correlation among features* is MEDIUM ∨ *correlation among features* is MED-HIGH ∨ *correlation among features* is HIGH) → SELECT	When the number of features is large and strongly correlated then feature selection should be used to remove less important features leading to reduced data size. Thus, reducing response time and improving accuracy
Feature Generation	If (# *of features* is LOW ∨ # *of features* is MED-LOW) ∧ (% of categorical features is LOW ∨ % of categorical features is MED-LOW) → SELECT	Small number of features makes it challenging for ML model to accurately classify the instances, thereby, new features should be generated from existing features especially if majority of the features are numerical which supports the generation of new features
Feature Transformation	If (*variance among features* is MED-HIGH ∨ *variance among features* is HIGH) ∧ (% of categorical features is LOW ∨ % of categorical features is MED-LOW) → SELECT	The large variation among values for the features makes it challenging for ML model to accurately classify the data and the chances of such a variation is higher in cases where majority of the features are numerical.
ML Algorithm Selection	If (*train sample size* is LOW ∨ *train sample size* is MED-LOW ∨ *train sample size* is MED) ∧ (# *of features* is LOW ∨ # *of features* is MED-LOW) → SELECT	If data has large number of features, most of the ML models work effectively. However, if the number of features is low and at the same time train data size is small, multiple ML algorithms should be tried before locking one specific algorithm.
Parameter Tuning	If (*variance among features* is MED ∨ *variance among features* is MED-HIGH ∨ *variance among features* is HIGH) ∧ (*train sample size* is LOW ∨ *train sample size* is MED-LOW ∨ *train sample size* is MED)	Parameter tuning is an important step, however, if the train data size is too large, the repetitive tuning of parameters becomes computationally expensive. Therefore, it should be used with small train data size especially if the variance among features is on the higher side.
Signature-based Detection	If (# *of features* is LOW ∨ # *of features* is MED-LOW ∨ # *of features* is MED) ∧ (*test sample size* is LOW ∨ *test sample size* is MED-LOW ∨ *test sample size* is MED) → SELECT	If test data size is large, another phase of detection in addition to anomaly detection increases the response time. Furthermore, anomaly detection is very effective for data with large number of features, therefore, there is no need of signature-based detection
False Positive Reduction	If (*test sample size* is LOW ∨ *test sample size* is MED-LOW) ∧ *variance among instances* is MED-HIGH ∨ *variance among instances* is HIGH) → SELECT	This component reduces false positive but at the cost of response time due to extra processing. Therefore, it should be used when test data size is small and variance among features is high as with low variance anomaly detection is already effective to generate few false positives
Alert Correlation	If (*test sample size* is MED-HIGH ∨ *test sample size* is HIGH) ∧ (# of features is MED-HIGH ∨ # of features is MED-HIGH) → SELECT	Large number of features offers the flexibility to correlate the alerts. Alert correlation is computationally expensive; therefore, it should be used only when then test data size is on the smaller side.
Alert Ranking	If (*variance for source bytes* is MED-HIGH ∨ *variance of source bytes* is HIGH) ∧ (*variance for destination bytes* is MED-HIGH ∨ *variance for destination bytes* is HIGH) → SELECT	Large variance for source bytes and destination bytes means the alerts are of diverse nature specified by the amount of alert embodied and so they should be ranked to prioritize response to more dangerous alerts

6.2. The membership of *number of features* is 0.2 for MED-LOW, 0.8 for MED, and 0 for other subsets. The inferences drawn using membership values and rule matrix are presented in Figure 6.3. Only two inferences are non-zero with one for SELECT and one for NOT SELECT. We used ROOT-SUM-SQUARE method (Equation 6.2) to combine strengths of non-zero inferences, which gives us a value of 0.2 for SELECT and 0.82 for NOT SELECT. Finally, we use Equation 6.3 to defuzzify the inferences' strengths with centroid values of 0 and 1 for SELECT and NOT SELECT decisions respectively. Equation 6.3 gives us output value 0.81. The value indicates that the signature-based detection component should not be selected with a confidence level of 81% for CIDDS.

Number of Features \ Test Sample Size	LOW	MED-LOW	MED	MED-HIGH	HIGH
LOW	S	S	NS	NS	NS
MED-LOW	S	S	NS	NS	NS
MED	NS	NS	NS	NS	NS
MED-HIGH	NS	NS	NS	NS	NS
MED-HIGH	NS	NS	NS	NS	NS

Figure 6. 1 Rule matrix for Selecting (S)/Not Selecting (NS) Signature-based detection component

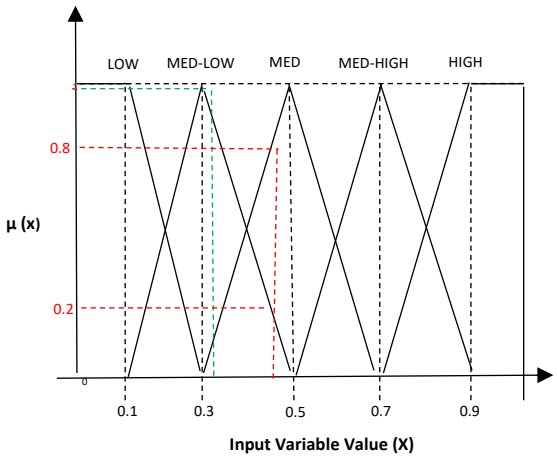


Figure 6. 2 Fuzzy subsets and degree of membership of the two input variables (i.e., Test sample size and Number of features) for Signature-based detection component

Test Sample Size		Number of Features	Use Minimum
If MED-LOW	and	LOW	→ **SELECT** → min (1, 0) = 0
If MED-LOW	and	MED-LOW	→ **SELECT** → min (1, 0.2) = 0.2
If MED-LOW	and	MED	→ **NOT SELECT** → min (1, 0.8) = 0.8
If MED-LOW	and	MED-HIGH	→ **NOT SELECT** → min (1, 0) = 0
If MED-LOW	and	HIGH	→ **NOT SELECT** → min (1, 0) = 0
If LOW	and	MED-LOW	→ **SELECT** → min (0, 0.2) = 0
If MED	and	MED-LOW	→ **NOT SELECT** → min (0,0.2) = 0
If MED-HIGH	and	MED-LOW	→ **NOT SELECT** → (0, 0.2) = 0
If HIGH	and	MED-LOW	→ **NOT SELECT** → (0, 0.2) = 0
If LOW	and	MED	→ **NOT SELECT** → min (0, 0.8) = 0
If MED	and	MED	→ **NOT SELECT** → min (0, 0.8) = 0
If MED-HIGH	and	MED	→ **NOT SELECT** → min (0, 0.8) = 0
If HIGH	and	MED	→ **NOT SELECT** → min (0, 0.8) = 0

Figure 6. 3 Fuzzy inferences for the selection of signature-based detection component

$$Membership,\ \mu(X) = \sqrt{rule1^2 + rule2^2 + - - - + ruleN^2} \tag{6.2}$$

$$Output = \frac{\sum_{i=1}^{N}(Center_i.Strength_i)}{\sum_{i}^{N} Strength_i} \tag{6.3}$$

6.2.5 System Architecture

The architecture of the system comprises of two major parts (i.e., managed system and adaptation controller) as shown in Figure 6.4. The managed system has already been discussed in Chapter 5. The

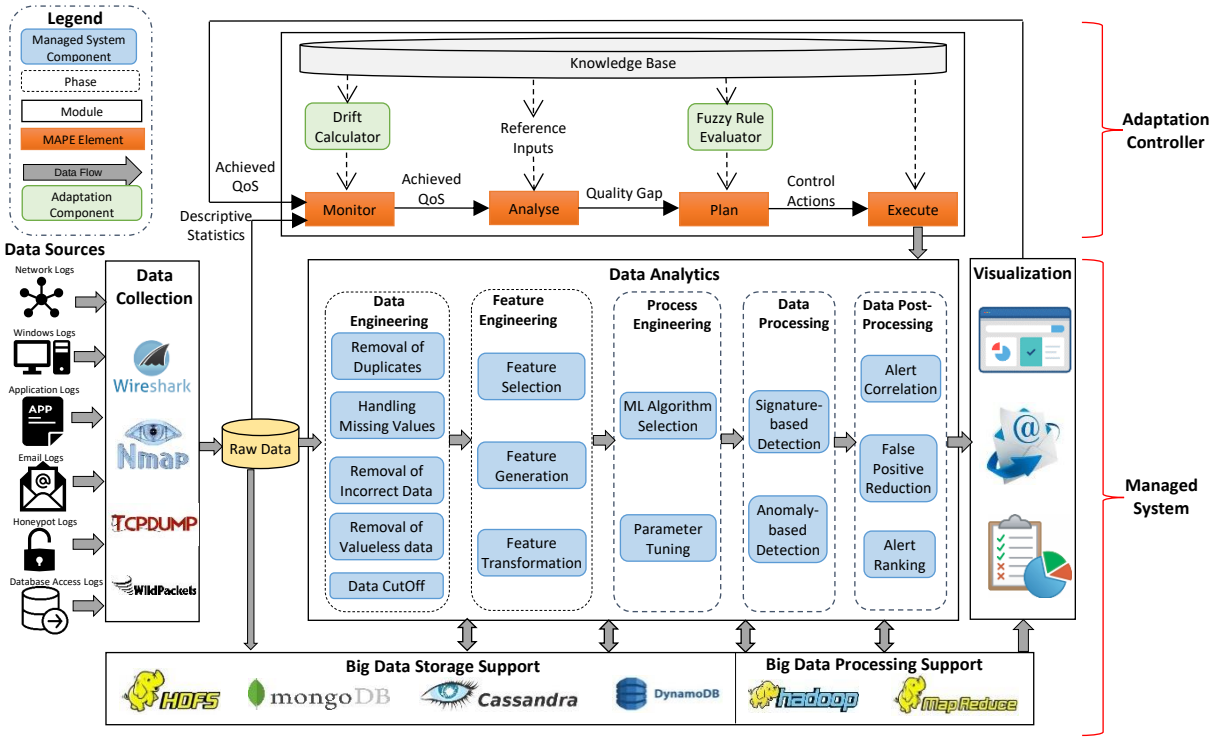


Figure 6. 4 BDCA architecture with adaptation controller for realizing adaptation

adaptation controller leverages MAPE-K pattern [219]. The *monitor* element uses drift calculator to monitor and compute the divergence D in the incoming security data (i.e., raw data) before feeding the data into the system. The divergence values are passed on to the *analyzer* element, which applies the *p-value* test to determine the significance of divergence. If the divergence is significant, the *analyzer* element triggers the *planner* element indicating to the *planner* that the BDCA system requires adaptation. The *planner* element uses fuzzy rule evaluator to first compute the values for the 13 input variables from the input data. Based on the values for input variables, the *planner* then applies the predefined fuzzy rules to decide which components should be selected for the new configuration of the system. After selecting the components, the information is forwarded to the *executor* that realizes the new configuration. The *knowledge base* maintains the information required for adaptation such as reference inputs and fuzzy rules.

6.3 Evaluation

In this section, we first present the adaptation scenarios, configurations, and data drift. We then present the evaluation objectives followed by evaluation results. The implementation setup for evaluating *QuickAdapt* is the same as reported in Section 5.4 for the evaluation of *ADABTics*.

6.3.1 Initial configuration and adaptation scenarios

As described in Section 6.2.2, *QuickAdapt* requires initial configuration of target QoS and weights. For target QoS, we insert best possible values for all metrics (e.g., 100% accuracy and 0% False Positive Rate) to Equation 5.1, which gives us a value of 0 for target QoS. We assigned a weight of 1 to all metrics in Equation 5.1 to specify the same level of significance. We evaluate our approach in two scenarios. *Baseline:* Here, the BDCA system is processing a particular data type (such as KDD dataset) with the *best* configuration for the specific data type and there is no change in the input data. *Change in input data:* Under this scenario, the data input into the system is changed (e.g., from KDD to DARPA). With

such a change, the monitor element of the adaptation controller (which constantly monitors the data input) signals the drift calculator, which computes the drift to determine whether or not the drift is significant according to the *p-value*. In case of a significant drift, adaptation is triggered.

6.3.2 Data drift

As mentioned in Section 6.2.3, we used KS test [217] to calculate the data drift (i.e., change in input data). The six data shifts, associated divergence D, and *p-values* are shown in Table 6.3. As an example, the KS test plot is shown for KDD→CIDDS in Figure 6.5, which underlines the distribution of the two datasets. The divergence D is same for shift in both directions e.g., KDD→DAPRA and DAPRA→KDD. The *p-value* is less than 0.05 for all cases, which specifies a significant change in data. Hence, adaptation is triggered in all considered cases.

Table 6. 3 KS test statistics for various data drifts

Data Shift	Divergence D	p-value
KDD→DARPA / DARPA→KDD	0.71	0.004
KDD→CIDDS / CIDDS→KDD	0.58	0.003
KDD→CICIDS2017 / CICIDS2017→KDD	0.38	0.001
DARPA→CIDDS / CDDS→DARPA	0.21	0.007
DAPRA→CICIDS2017 /CICIDS2017→DARPA	0.29	0.019
CIDDS→CICIDS2017 / CICIDS2017→CIDDS	0.32	0.014

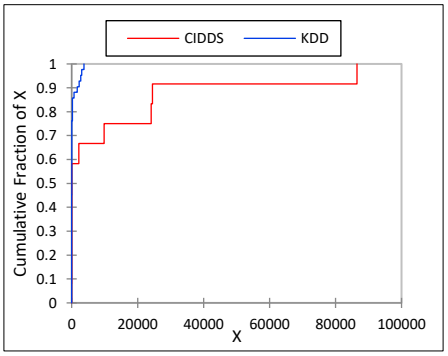


Figure 6. 5 KS test plot for drift from KDD to CIDDS

6.3.3 Evaluation objectives

The evaluation of *QuickAdapt* aims to answer the following Research Questions (RQ).

- ***RQ1:*** *What is the adaptation accuracy of QuickAdapt?*
- ***RQ2:*** *How long does it take for QuickAdapt to make the system adapt to the changes in data (i.e., adaptation time)?*
- ***RQ3:*** *How much optimization is achieved in accuracy and prediction time using QuickAdapt (i.e., optimization)?*
- ***RQ4:*** *How QuickAdapt performs in comparison to ADABTics [5]?*

6.3.4 Evaluation results

6.3.4.1 RQ1: Adaptation accuracy

Adaptation accuracy measures the ability of *QuickAdapt* to select components that result in a configuration with a high QoS. The possible combinations of components considered in this study

result in 32 configurations shown in Table 5.1. We measure the adaptation accuracy using top-5 accuracy and average accuracy proposed in [180]. In our case, the top-5 accuracy means whether the configuration selected by *QuickAdapt* is included in the configurations with the top 5 achieved QoS. On the other hand, an average accuracy measures whether the selected configuration has a QoS above the average QoS for the 32 configurations. Table 6.4 presents the configurations selected for the four datasets and the achieved QoS for the 20 experimental cases (i.e., 5 ML algorithms × 4 datasets). It is important to note that the lower the value of QoS, the better is the QoS). Figure 6.6 shows the achieved QoS for the 32 configurations for the 20 experimental cases. According to the top-5 measure, the **adaptation accuracy of our approach is 70%**. This means that for 14/20 cases; the selected components and the resulting configuration are included in the top-5 configurations. The adaptation accuracy of our approach is 85% according to the average measure, which implies that for 17/20 cases the selected configuration has an above average QoS. The top-5 accuracy is 60% (3/5) for KDD, 80% (4/5) for DAPRA, 80% (4/5) for CIDDS, and 60% (3/5) for CICIDS2017. The components in each of the selected configurations are in accordance with the rules presented in Table 6.2. For example, the higher percentage of duplicate records in KDD and CIDDS (i.e., 78% and 40.5% respectively) results in the selection of Removal of Duplicates component for KDD and CIDDS datasets.

6.3.4.2 RQ2: Adaptation time

Adaptation time measures the speed with which *QuickAdapt* can adapt a system to the change in input data. Adaptation time is counted from the moment adaptation is triggered to the moment the system gains the desired state *[219]*. The time taken by *QuickAdapt* to enable the system to adapt to each of the

Table 6. 4 Achieved QoS for various datasets and Machine Learning algorithms

Dataset	Selected Configuration	K-means	XGBoost	Naïve Bayes	SVM	Random Forest
KDD	C1-C3-C4-C5	1.31	0.86	0.61	0.25	0.71
DARPA	C4-C5	0.99	0.59	0.72	0.78	2.01
CIDDS	C1-C5-C6	0.61	0.55	0.66	0.48	1.1
CICIDS2017	C2-C3-C5-C6	0.36	1.05	0.21	1.33	0.18

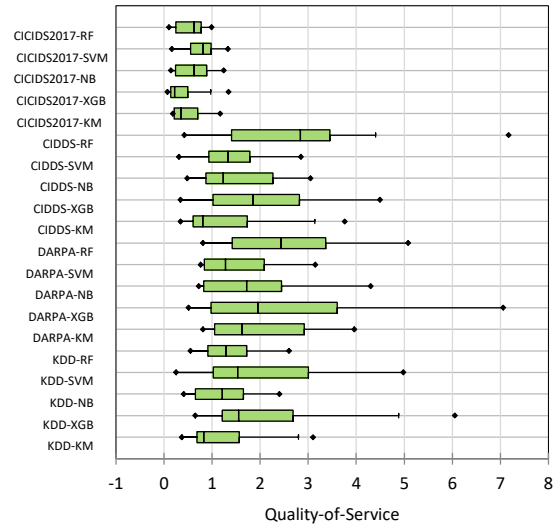


Figure 6. 6 QoS values for 32 configurations in each of the 20 cases (5 ML algorithms × 4 datasets)

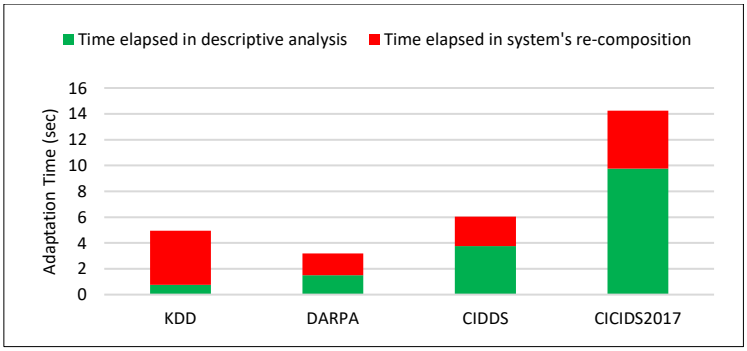


Figure 6. 7 Adaptation time with *QuickAdapt* for the four datasets

four datasets is shown in Figure 6.7. On average, ***QuickAdapt* takes around 7.1 sec to adapt the system**. The time is primarily elapsed in the descriptive analysis of a dataset and re-composition of the system i.e., adding and removing components. On average, 55.36% adaptation time in elapsed in descriptive analysis while 44.63% is elapsed in re-composition. The descriptive analysis time is longest (i.e., 9.75 sec) for CICIDS2017 followed by CIDDS (3.75 sec), DARPA (1.5 sec), and KDD (0.75 sec). From this trend, we assert that the descriptive analysis time relates to the number of features and number of instances in the dataset. CICIDS2017 having 81 features and 655,910 instances took the longest time while KDD having 42 features and a comparatively small number of instances (i.e., 311,029) took the shortest time. The time elapsed in re-composition is 4.5 sec for CICIDS2017, 4.2 sec for KDD, 2.3 for CIDDS, and 1.7 sec for DARPA. This trend is in line with the number of components invoked for each dataset as presented in Table 6.4 i.e., four components (C1, C3, C4, C5) for KDD, two components (C4, C5) for DARPA, three components (C1, C5, C6) for CIDDS, and four components (C2, C3, C5, C6) for CICIDS2017 The invocation of four components for CICIDS2017 and KDD took longer as compared to three and two components invocation for CIDDS and DARPA respectively.

6.3.4.3 RQ3: Optimization

We assess optimization by comparing the accuracy and prediction time of the system exactly before and after adaptation. Table 6.5 reports the accuracy and prediction time for the four datasets and five algorithms before and after adaptation. On average, ***QuickAdapt* improves accuracy and prediction time by 4.52% and 17.6% respectively**. With respect to datasets, the accuracy improves by the largest margin for KDD and CICIDS2017 (5.1%) followed DARPA (4.2%), and CIDDS (3.7%). The same trend is observed for prediction time with KDD benefiting the most (i.e., 21.4%) followed by CICIDS2017 (18.5%), DARPA (15.4%), and CIDDS (15.3%). A reason for the significantly larger impact for KDD and CICIDS2017 is the inclusion of the feature selection component as illustrated in Table 6.4. Given the comparatively large number of features in both datasets (i.e., 41 features in KDD and 82 features in CICIDS2017), the feature selection component selects the most relevant features, which helps in the improvement of accuracy and prediction time. In order to assess whether the achieved improvement is statistically significant, the values before and after adaptation are subjected to Wilcoxon's signed ranked test [198] at a significance level $\alpha = 0.05$. The *p-values* for accuracy and prediction time are 0.004 and 0.0001 respectively. Since both *p-values* are lower than 0.05, we can conclude that *QuickAdapt* significantly improves both accuracy and prediction time of a BDCA system.

6.3.4.4 RQ4: Comparison with ADABTics

We compare *QuickAdapt* with *ADABTics* [5] (presented in Chapter 5) in terms of adaptation time, adaptation accuracy, and level of optimization. Table 6.6 compares the time taken by the two

approaches to adapt the system to various datasets. On average, **adaptation time for *QuickAdapt* is 105 times lower than *ADABTics***. This is because *ADABTics* executes all configurations before selecting

Table 6. 5 Accuracy and prediction time of the BDCA system before and after adaptation

Dataset	ML Algorithm	Accuracy (%)		Optimization (%)	Prediction Time (sec)		Optimization (%)
		Before Adaptation	After Adaptation		Before Adaptation	After Adaptation	
KDD	Kmeans	82	91.6	9.6	17.5	13.4	23.4
	XGBoost	87.1	90.4	3.3	11.4	8.7	23.6
	Naives Bayes	89.3	92.4	3.1	23	21	8.6
	SVM	87	91.7	4.7	20.4	15.8	22.5
	Random Forest	88.1	92.6	4.5	19.5	13.9	28.7
DARPA	Kmeans	78.1	81.4	3.34	21.7	20.4	5.9
	XGBoost	80.4	85.7	5.3	23.8	18.2	23.5
	Naives Bayes	81.1	83.4	2.3	52.6	45.7	13.1
	SVM	79.8	84.5	4.7	34.7	29.2	15.8
	Random Forest	83.5	89	5.5	27.6	22.4	18.8
CIDDS	Kmeans	76.8	80.5	3.7	34.7	29.8	14.1
	XGBoost	74.6	78.7	4.1	24.7	19.6	20.6
	Naives Bayes	75.7	79.5	3.8	24.6	24.1	2.1
	SVM	81.4	83.7	2.3	16.7	13.5	19.1
	Random Forest	78.4	83.1	4.7	15.6	12.4	20.5
CICIDS2017	Kmeans	85.6	93.4	7.8	14.7	12.9	12.2
	XGBoost	87.4	92.9	5.5	11.6	8.4	27.5
	Naives Bayes	85.5	90.1	4.6	22.6	19.8	12.3
	SVM	89.5	93.1	3.6	27.6	21.4	22.4
	Random Forest	88.6	92.7	4.1	23.7	19.4	18.1

the best configuration while *QuickAdapt* does not rely on executing configurations rather selects a configuration based on fuzzy rules and descriptive analysis of the data. Since *ADABTics* calculates and subsequently compares the QoS delivered by each configuration under the Most Optimal QoS condition, it always selects the configuration with the best QoS. Hence, adaptation accuracy for *ADABTics* is 100% as compared to the 70% adaptation accuracy for *QuickAdapt*. Figure 6.9 compares the optimization achieved in accuracy and prediction time by both approaches. On average, *ADABTics* optimizes accuracy by 6.2% as compared to 4.5% optimization achieved by *QuickAdapt*. The average optimization in prediction time is 26.1% for *ADABTics* and 17.6% for *QuickAdapt*. The optimization comparison is aligned with the adaptation accuracies for the two approaches. *ADABTics* always selects a configuration with the best QoS, which is calculated based on accuracy and prediction time as evident from Equation 5.1. Therefore, the optimization achieved with *ADABTics* is higher than that with *QuickAdapt*. Whilst the adaptation accuracy for *QuickAdapt* is lower than *ADABTics*, the gain in adaptation time (i.e., 105 times) for *QuickAdapt* is far significant than the marginal loss (i.e., 30%) in adaptation accuracy.

Table 6. 6 Comparison of adaptation time for *QuickAdapt* and *ADABTics*

Dataset	**ADABTics**	**QuickAdapt**
KDD	545.30	4.95
DARPA	651.80	3.20
CIDDS	736.00	6.05
CICIDS2017	1073.00	14.25

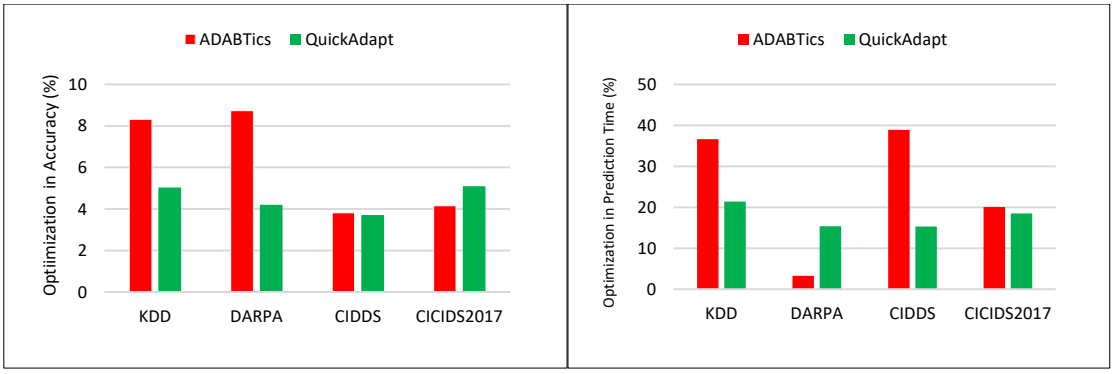


Figure 6. 8 Comparison of optimization in accuracy and prediction time with *QuickAdapt* and *ADABTics*

6.4 Threats to Validity

Internal Validity: The fuzzy rules presented in Table 6.2 are formulated based on a systematic literature review presented in Chapter 2 and experiences from large scale experimentation on BDCA presented in Chapter 3. However, we do not claim the perfection of these rules. We believe that the rules can further be refined, and more rules can be added to fine tune the component selection process. ***Construct Validity:*** We evaluated *QuickAdapt* with the change in input data from one dataset to another (e.g., KDD to DARPA). The divergence for such a drift is quite high and so the indication for triggering adaptation is very clear. The adaptation triggering capability of *QuickAdapt* can be challenged when the changes in input data are slow and steady. ***External Validity:*** The six components considered in our experimentation can be implemented in different ways. Therefore, our specific implementation of the components can also pose a threat to the generalization of the findings reported in this chapter.

6.5 Related Work

In this section, we position the novelty of the work reported in this chapter with respect to related studies.

Several studies (e.g., [5, 214-216, 223]) have explored adaptation for QoS optimization. Calinescu et al., [214] use KAMI model based on Bayesian estimator to estimate model parameters for adapting a service-based system. The approach proposed in [215] uses a hierarchical model for choosing network QoS-parameters (e.g., network bandwidth and network delay) to adapt mobile computing system. Cardellini et al., [216] use linear programming to realize adaptation in a service-oriented architecture for achieving optimal QoS. Edwards et al., [223] propose a silo approach of monitoring each component of a robotic system and triggering adaptation when a component's performance degrades. Whilst [214-216, 223] are in different domains using different adaptation techniques, the work that is closest to ours is *ADABTics* [5] (reported in Chapter 5), where a BDCA system is adapted to changes based on deviation in QoS from a predefined threshold. Unlike prior works, *QuickAdapt* is the first of its kind that leverages fuzzy rules and descriptive statistics of the input data to adapt a BDCA system. Since *ADABTics* is the only work in the domain of BDCA using similar ML algorithms and datasets, we have directly compared *QuickAdapt* with *ADABTics* in Section 6.3.4.4.

ADABTics, presented in Chapter 5, takes around 751.5 sec on average to find the optimal configuration from the same search space of 32 configurations also considered in our study. Several recent studies [224-226] also faced the same issue of large adaptation time and proposed solutions accordingly. Alipourfard et al., [224] employed Bayesian optimization to reduce the adaptation time by up to 75%.

Hill et al., [225] use hill climbing to select an optimal configuration at runtime, which reduces the adaptation time by around 50%. Jiang et al., [226] exploit spatial and temporal correlations among various configurations to quickly select the optimal configuration for video analytics, hence, reducing adaptation time by 2-3 times. In addition to being in a different domain, *QuickAdapt* also differs from the previous works with respect to the optimization principle. The previous works aim to optimally search and subsequently select a configuration from the available configurations, which is a time costly operation. Instead of searching, *QuickAdapt* selects components based on predefined fuzzy rules that ultimately generate the optimal configuration. Hence, reducing the adaptation time by 105 times in comparison to *ADABTics* as demonstrated in Section 6.3.4.4.

6.6 Chapter Summary

In this chapter, we argued that the accuracy and prediction time of a BDCA system degrade with changes in security events data. Hence, a BDCA system needs to be adapted with the changes. However, the adaptation time should not surpass the gain in prediction time achieved through adaptation. We presented *QuickAdapt*, a time-efficient adaptation approach that uses descriptive statistics of security events data and fuzzy rules to establish a relationship between the input security data and component-based architecture of a BDCA system. The adaptation approach leverages the fuzzy rules to operationalize components that are most optimal for the particular data type (e.g., KDD and CICIDS2017). Through rigorous empirical evaluation, we demonstrate that *QuickAdapt* reduces the adaptation time by a factor of 105 (i.e., 105×) with minimal impact on the adaptation accuracy.

Chapter 7

On the Scalability of Big Data Cyber Security Analytics Systems

Related publication:

This chapter is based on our paper, which is under review in the journal of *Future Generation Computer Systems (FGCS 2020)* "On the Scalability of Big Data Cyber Security Analytics Systems" [3].

Chapter 2 reveals response time, accuracy, and scalability as the three most important quality attributes of a BDCA system. Among the 74 BDCA papers reviewed in Chapter 2, 40 papers emphasized the importance of scalability for a BDCA system. Despite the high emphasis, there has been little effort to investigate the scalability of a BDCA system to identify and exploit the sources of scalability improvement. Therefore, in this chapter, we first investigate the scalability of a BDCA system. We then identify the factors that significantly impact the scalability of the BDCA system. Based on the identified factors, we finally propose a parameter-driven adaptation approach, *SCALER*, for optimizing the scalability of the BDCA system. For investigating and optimizing scalability, we conducted experiments by implementing a Spark-based BDCA system on a large-scale OpenStack cluster. We run our experiments with four security datasets presented in Chapter 3. The findings of this chapter highlight the importance of exploring and exploiting the configuration parameter space of the underlying big data framework (e.g., Apache Spark) for scalable security analytics.

7.1 Introduction

Chapter 2 discloses the 12 most important quality attributes of a BDCA system, where scalability is ranked as the third most important quality attribute of a BDCA system. Scalability is defined as "*the system's ability to increase speedup as the number of processors (nodes) increase*" [227]. The rationale behind a BDCA system being highly scalable is twofold – (a) the volume of security event data is rapidly increasing, which requires a BDCA system to scale up (by adding more computational power) to process the data without impact on the response time of the system [35, 36] and (b) the velocity with which the security event data fluctuates over time [23, 228]. For example, a BDCA system analysing network work traffic of a bank experiences a higher workload during working hours as compared to non-working hours. Therefore, the BDCA system should efficiently use the commodity or third-party resources to scale out (i.e., add nodes) during working hours and scale in (i.e., remove nodes) during non-working hours. Whilst scaling out and scaling in are inevitable for a BDCA system, it is important that a BDCA system takes maximum benefit from the addition of the resources during the scale out process to reduce the operational cost. For instance, a BDCA system utilizing an additional

computational node merely up to 40% is not benefitting in proportion to the amount an enterprise pays for the addition of the node.

Among the 74 studies on BDCA reviewed in Chapter 2, 40 studies highlight the importance of scalability for a BDCA system. Among the 40 studies, some studies (e.g., [28, 46, 153, 229, 230]) explore how a BDCA system scales as more number of computing nodes are added. Moreover, [28] presents an approach based on hot spot migration [113] to balance workload among the nodes and [46] monitors the workload to decide whether data should be processed locally or burst to another cluster. However, none of the studies have either investigated the factors that impact BDCA system's scalability nor have devised any solutions for improving the scalability. Several BDCA studies (e.g., [45, 231, 232]) hint at factors such as machine learning algorithm employed in the system for anomaly detection, quality of security event data, and big data processing framework that can potentially impact the scalability. Among these factors, the most prominent is the underlying big data processing framework (e.g., Spark and Hadoop), which is an integral part of any BDCA system. One of the core features of any big data processing framework is its configuration parameters (e.g., executor memory) [51], which guides how the framework should process the data. For example, executor memory specifies how much memory should be allocated to the executor process. The importance of parameter configuration for big data processing frameworks has been highlighted by several studies (e.g., [35, 233, 234]). However, none of the previous studies have investigated their impact on the scalability of any big data system including BDCA system. Given that a BDCA system can be architected on top of several big data frameworks such as Spark and Hadoop, we considered a Spark-based BDCA system in this study. This is because, unlike the works reported in the last four chapters, this work is focussed on investigating and exploiting the underlying configuration parameters of the big data framework for scalable analytics. In comparison to other big data frameworks especially Hadoop, Spark offers far many parameter optimizations options such as controlling utilization of memory bandwidth, garbage collection, and number of disk processer per second [235]. This is why several studies (e.g., [236-240]) also select Spark for investigating parameter optimization as compared to a few studies (e.g., [241, 242]) that select Hadoop. Therefore, in this chapter, we aim to "***investigate the impact of Spark configuration parameters on the scalability of a BDCA system and devise an automated solution for parameter configuration to optimize scalability***.

Chapter Contribution: In order to achieve the aforementioned aim, in this chapter, we contribute by answering the following research questions.

RQ1: *How a BDCA system scales with default Spark Configuration settings?*

RQ2: *What is the impact of tuning Spark Configuration Parameters on the scalability of a BDCA system?*

RQ3*: How to improve the scalability of a BDCA system?*

To answer the three research questions, we developed an experimental infrastructure on a large-scale OpenStack cloud. We then implemented a Spark-based BDCA system that runs on the OpenStack cloud in a fully distributed fashion. For measuring scalability, we defined our own scalability metric reported in Section 7.2. Similar to the previous chapters, we used four security datasets (i.e., KDD [55], DARPA [56], CIDDS [57], and CICIDS2017 [58]) for scalability investigation. Based on our comprehensive experimentation, we found that:

(i) A BDCA system with default Spark configuration parameters does not scale ideally. The deviation from ideal scalability is around 58%. This means a system only takes 42% benefit made available by the additional resources.

(ii) Among the 11 investigated Spark parameters, changing the value of nine parameters significantly impacts a BDCA system's scalability. The optimal value of a parameter (with respect to scalability) varies from dataset to dataset.

(iii) We proposed and evaluated a parameter-driven adaptation approach, *SCALER*, that automatically selects the most optimal value for each parameter at runtime. The evaluation results show that on average, *SCALER* improves a BDCA system's scalability by 16%.

Chapter Organization: Section 7.2 describes our BDCA system, the scalability metric, and our parameter-driven adaptation approach for optimizing scalability. Section 7.3 presents the detailed findings of the chapter with respect to the three research questions. Section 7.4 presents our reflections on the findings. Section 7.5 outlines the threats to the validity of the findings presented in this chapter. Section 7.6 positions the novelty of the work presented in this chapter with respect to other related studies. Finally, Section 7.7 concludes the chapter highlighting the implications of the findings both for practitioners and researchers.

7.2 Methods

This section describes our BDCA system, the instrumentation setup, scalability metric, and our adaptation approach (*SCALER*) for optimizing scalability.

7.2.1 Our BDCA system

The BDCA system used for experimentation in this study is depicted in Figure 7.1. Our BDCA system consists of three layers – *Security Analytics Layer*, *Big Data Support Layer*, and *Adaptation Layer*. In the following, we describe Security Analytics Layer and Big Data Support Layer while the details of Adaptation Layer are presented in Section 7.2.5.

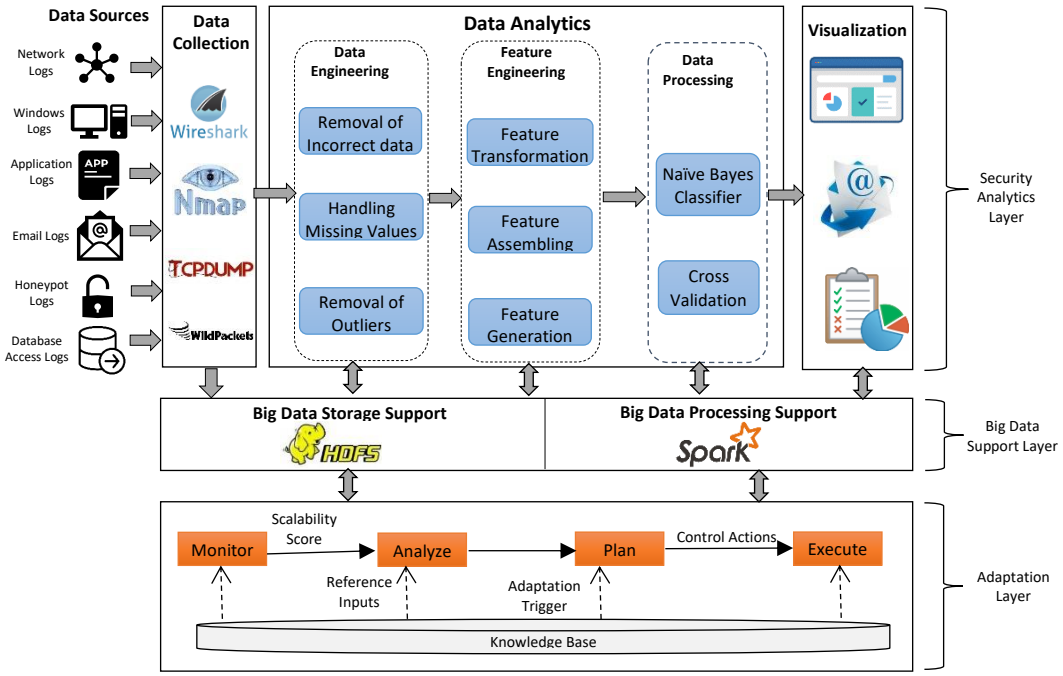


Figure 7. 1 BDCA system considered for experimentation in this study

7.2.1.1 Security analytics layer

Security analytics layer processes the security event data for detecting cyber-attacks. Unlike the BDCA systems considered in the previous chapters, the BDCA system considered in this chapter consists of three phases (i.e., data engineering, feature engineering, data processing) in the security analytics layer. This is because this chapter studies the impact of big data framework configuration unlike the previous chapters that study the impact of architectural configuration. That is why we considered a rather simple BDCA system for experimental purposes in this chapter.

Data Engineering pre-processes the data to handle missing values and remove incorrect values and outliers. A negative value indicates that the number of features for the instances are incomplete, hence, the instance is removed. Incorrect values (e.g., standard deviation = -1) in the dataset are unacceptable for the Machine Learning (ML) model employed in the system for classification of security event data into normal and attack classes. Therefore, we use the filter method of the DataFrame available in the Spark package, i.e., *org.apache.spark.sql* to remove the incorrect values. The existence of the outliers in the training dataset affects the accuracy of the machine learning model. We, therefore, removed the values that were larger than *Double.MaxValue*. CICIDS2017 dataset has lots of missing values, therefore, we removed the instances with the missing values by simply investigating whether the 82nd value (last feature) is negative.

Feature Engineering generates new features, and/or transforms the values of features into a new range. For all four datasets, we assembled the features to transform multiple columns of features into one column of feature vector for fitting the ML model. We used the *VectorAssembler* method in the *org.apache.spark.ml.feature* for the implementation of assembling the features. Since some algorithms (e.g., Naïve Bayes) in the SparkML library [53] cannot handle non-numeric features, we used *StringIndexer (from org.apache.spark.ml.feature*) to transform the label features (i.e., normal and attack) in KDD dataset from string to indices. Given the relatively smaller number of features in the DARPA dataset, we expanded the features to a polynomial space. We used the *PolynomialExpansion* method in the *org.apache.spark.ml.feature* for feature expansion.

Data Processing leverages a machine learning algorithm to classify the instances in the security data as either *normal* or *attack*. In our system, we used Naïve Bayes classifier for classifying the instances. We choose Naïve Bayes algorithm because it is a widely used algorithm in BDCA domain that can achieve good classification accuracy when the data size is large, and the features are independent of each other [1]. Furthermore, Naïve Bayes is easy to update and leads to linear time for classification [243]. We used the Spark package *org.apache.spark.ml.classification* for implementing the Naïve Bayes model. For cross-validation of the model, we used the *CrossValidator* method available in the *org.apache.spark.ml.tuning*.

7.2.1.2 Big data support layer

Big data support layer supports the security analytics layer by managing the distributed storage and processing of data on multiple computing nodes. The layer consists of big data processing framework (i.e., Spark) and big data storage (i.e., HDFS). Apache Spark is a an open-source big data processing framework that uses in-memory primitives to process large amount of data [244]. Spark is quite suitable for machine learning tasks, which requires iterative processing that best suits Spark architecture [244]. Moreover, Spark is compatible with multiple file systems such as HDFS, MongoDB, and Cassandra. Hadoop Distributed File System (HDFS) is a data storage system that enables the

distributed storage of massive amount of data [54]. By default, HDFS replicates each block of data on three nodes, which makes it highly fault tolerant.

7.2.2 Instrumentation setup

We configured Spark and Hadoop (for HDFS) on an OpenStack cluster consisting of 10 computing nodes. Each node is installed with Ubuntu 16.04 Xenial Xerus operating system. Each node runs Spark 2.4.0, Hadoop 2.9.2, and JDK 1.8. The 10 computing nodes are divided into master and slave nodes. There is one master with *m1.large* flavour (4 GB RAM, 80 GB Hard disk, and 8 VCPUs) and nine worker nodes with *m1.small* flavour (2 GB RAM, 10 GB Hard disk, and 1 VCPU). Each node in the cluster has a floating IP for communication with the external world and an internal IP for communicating with other nodes in the cluster. To associate floating IP with internal IP, a router is created as the bridge between external network (floating IPs) and subnets (internal IPs). We used Scala programming language for various implementations (Section 7.2.1) on Spark. For scalability investigation and evaluation of *SCALER*, we used the same four datasets (i.e., KDD [55], DARPA [56], CIDDS [57], and CICIDS2017 [58]) as used in the studies reported in the previous chapters. The details of the datasets are available in Chapter 3.

7.2.3 Scalability metric

Scalability metric assesses to what extent our BDCA system takes advantage of the additional hardware resource added to the system in the form of computing nodes. Several studies (e.g., [245-247]) have proposed metrics for measuring scalability of a system. However, the previous metrics are not applicable for the scalability analysis in our study for two reasons. First, the previous metrics do not quantify scalability with respect to ideal scalability, which is required for evaluating the effectiveness of our adaptation approach. Second, the previous metrics are primarily for measuring scalability in cases where the system is partly executed in parallel mode and partly in sequential mode whereas our system is executed fully in parallel mode.

For this study, we use Equation 7.1 to measure scalability of our BDCA system. In Equation 7.1, *S(c)* denotes the scalability score for curve '*c*'. For example, Figure 7.2 shows eight curves that illustrate how the system scales in each of the considered scenarios. *Gap* denotes the quantified gap value between the achieved and ideal response time, which is calculated using Equation 7.2. In Equation 7.2, ω_n denotes the user-defined weight that specifies the importance of gap between achieved and ideal response time at '*n*' worker nodes. For example, ω_2 is the weight for specifying importance of gap at two worker nodes and ω_4 is the weight for specifying the importance of gap at four worker nodes. In Equation 7.1, ω_{10} denotes the weight for specifying the importance of gap at 10 nodes i.e., to specify the importance of gap beyond eight nodes. The sum of all weights is equal to 1 (as presented in Equation 7.5). In Equation 7.2, G_n denotes the ratio of unaccomplished response time improvement to the response time improvement in the ideal case with '*n*' worker nodes. G_n is calculated using Equation 7.3, where AT_n denotes achieved response time with '*n*' worker nodes and IT_n denotes the ideal response time with '*n*' worker nodes. In Equation 7.1, *Trend*, which is calculated using Equation 7.4, denotes how response time decreases from six to eight worker nodes, the higher of which indicates the probability that the response time tends to decrease with more than eight nodes.

We illustrate the use of the scalability metric with an example, which includes eight possible scalability scenarios. Table 7.1 presents the hypothetical response times for eight different scenarios with five

$$S(c) = 1 - Gap - \omega_{10} \times (1 - Trend) \quad (7.1)$$

$$\text{Gap} = \omega_2 \times G_2 + \omega_4 \times G_4 + \omega_6 \times G_6 + \omega_8 \times G_8 \quad (7.2)$$

$$G_n = \frac{AT_n - IT_n}{IT_1 - IT_n} \quad (7.3)$$

$$Trend = \frac{AT_6 - AT_8}{IT_6 - IT_8} \quad (7.4)$$

$$\omega_2 + \omega_4 + \omega_6 + \omega_8 + \omega_{10} = 1 \quad (7.5)$$

Table 7. 1 Response time (in sec) for the eight hypothetical scalability scenarios

Number of Worker Nodes	Response Time (sec)							
	Ideal Scenario	Scenario-1	Scenario-2	Scenario-3	Scenario-4	Scenario-5	Scenario-6	Scenario-7
1	8	10	10	8	8	9.5	8	8
2	4	11	6.875	5.5	7	5	7	8
4	2	6	5	4	6	8	7.2	8
6	1.33	4	3.75	3	5.8	5.5	6	8
8	1	3	3.2	2.9	5.7	6	7.2	8

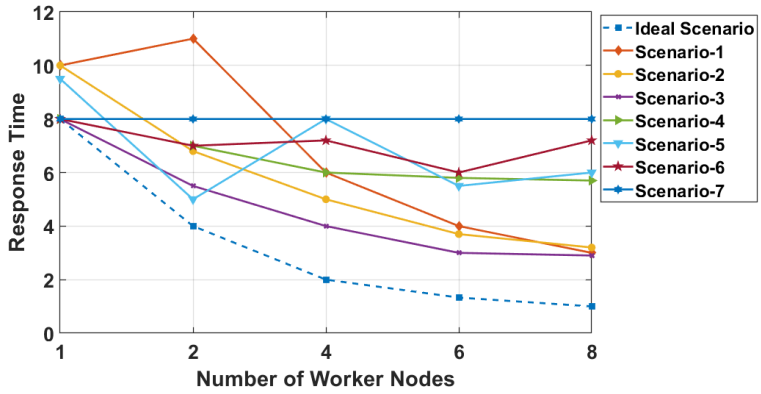


Figure 7. 2 Hypothetical scalability scenarios (drawn based on Table 7.1) to illustrate the use of scalability metric

different cluster configurations, i.e., 1 worker, 2 workers, 4 workers, 6 workers, and 8 workers. We use these five cluster configurations to illustrate the correctness of our metric. However, our metric an be customized for other cluster configurations. Figure 7.2 shows the eight scalability curves drawn using the response times reported in Table 7.1 Table 7. 1. Ideal Scenario underlines the case where each time the number of nodes is doubled, the response time is reduced to half. For calculating the scalability score, we use a value of 0.2 for all weights (i.e., ω_2 – ω_{10}). For calculating Trend in this scenario, AT_6 = 1.33, AT_8 = 1, IT_6 = 1.33, and IT_8 = 1 as shown in Table 7.1, hence, Trend = 1 using Equation 7.4. All gaps, i.e., G_2 – G_8 = 0 as there is no gap between achieved and ideal response time, hence, overall gap, i.e., Gap = 0 using Equation 7.3. Feeding these values into Equation 7.1 gives us S(ideal) = 1. For Scenario-1, AT_6 = 4, AT_8 = 3, IT_6 = 1.66, and IT_8 = 1.25, hence, Trend = 2.44, which is high – indicating a positive trend of scalability after eight worker nodes. The values of G_n calculated using Equation 7.3 are G_2 = 1.2, G_4 = 0.46, G_3 = 0.28, and G_8 = 0.2, which gives Gap = 0.42. Hence, the scalability score for Scenario-1 is 0.85, which indicates poor scalability as compared to ideal scalability. This is also observable from the comparison of the two curves, i.e., Ideal Scenario and Scenario-1 as depicted in Figure 7.2. As compared to Scenario-1, there is a higher reduction trend in response time with the increase in the number of nodes in the ideal case. Therefore, our scalability metric gives a smaller scalability score for Scenario-1 as compared to ideal scenario.

The scalability score for Scenario-2 is 0.83, which is slightly lower than the scalability score for Scenario-1. The slight difference is mainly due to the difference in Trend for the two scenarios, i.e., a reduction from 4 to 3 in Scenario-1 and a reduction from 3.75 to 3.2 in Scenario-2. The scalability score for Scenario-5 is 0.31, which is quite lower as compared to Scenario-1 and Scenario-2. If we observe the scalability curve for Scenario-4 in Figure 7.2, the response time reduces quite significantly as we increase the number of worker nodes from 1 to 4 nodes. However, there is almost no reduction as the number of worker nodes are increased from 4 to 8, which is why the scalability score is much lower as compared to more smoother curves such as the curves for Scenario-1 and Scenario-2. In Scenario-5, the sudden upward jump in the curve from 2 nodes to 4 nodes impacts the scalability score of the whole curve. Therefore, the scalability is quite low, i.e., 0.16. In Scenario-6, the response time increases (unlike expected) at two transitions, i.e., from 2 nodes to 4 nodes and from 6 nodes to 8 nodes. Therefore, the negative impact on the response time at two transitions significantly impact the scalability score and Equation 7.1 generate a much lower scalability score (i.e., -0.56) for Scenario-6. The response time in Scenario-7 does not change with the change in the number of nodes, therefore, the scalability score for Scenario-7 is 0.

7.2.4 Optimizing scalability with *SCALER*

In order to optimize the scalability of a BDCA system, we present *SCALER* - an adaptation approach that automatically triggers the tuning process and tunes Spark configuration parameters. By tuning, we mean to select a combination of parameters, which generates a scalability score that is above the predefined threshold.

Spark configuration parameters control most of the application settings and directly impact the way an application runs [248]. All Spark parameters have a default configuration; however, the default configuration is not suitable for each application [248]. Therefore, the parameters need to be configured separately for each application. The Spark parameters investigated in this study for their impact on scalability of the system are presented in Table 7.2. We selected 11 parameters (out of 187) based on the

Table 7. 2 Spark parameters considered in the study for their scalability impact and tuning

ID	Spark Parameter	Default Value	Description
P1	Spark.executor.memory	1g	Amount of memory used by each executor process in Spark
P2	Spark.shuffle.sort.bypassMergeThreshold	200	Underlines the threshold of reduce partitions for avoiding merge-sorting data
P3	Spark.shuffle.compress	TRUE	Whether (or not) to compress the map output files
P4	Spark.memory.storageFraction	0.5	Amount of memory available for task execution
P5	Spark.shuffle.file.buffer	32k	Amount of memory available to buffer file output streams
P6	Spark.reducer.maxSizeInFlight	48m	Maximum size of map output to be fetched for reducer
P7	Spark.memory.fraction	0.6	Proportion of heap size used for execution and storage
P8	Spark.serializer.objectStreamReset	100	To allow or stop garbage collection of objects
P9	Spark.rdd.compress	FALSE	Whether (or not) to compress Resilient Data Distributed Datasets (RDD)
P10	Spark.shuffle.memoryFraction	\ (Deprecated)	Proportion of Java heap size used for aggregation. Beyond this limit, contents start spilling to the disk
P11	Spark.driver.memory	1g	Amount of memory used for initializing Spark context

following criteria – (i) the parameters have proven impact on different aspects of Spark such as scheduling, compression, and serialization (ii) the parameters contribute to Spark running time as highlighted in [239] and [238] and (iii) the parameters impact multiple levels (e.g., machine level and cluster level) of a BDCA system as reported through industry practices [248, 249]. Although *SCALER* considers 11 Spark parameters, it is worth noting that SCALER can be easily extended to incorporate more parameters if needed.

In the following, we describe our adaptation approach (*SCALER*) that automatically tunes the parameters presented in Table 7. 2 for optimizing the scalability. We present our adaptation approach as per the guidelines for adaptation approaches presented by Villegas et al., [219].

7.2.4.1 Adaptation goal

The adaptation goal underpins the main reason for adaptation i.e., why a BDCA system needs to adapt [219]. Our adaptation goal is driven by the results collected for RQ1 (Section 7.3.1) and RQ2 (Section 7.3.2). These findings indicate that (i) with default settings of Spark parameters, a BDCA system does not scale ideally and (ii) nine out of 11 studied Spark parameters significantly impact the scalability of the BDCA system. Therefore, our adaptation approach aims to "*optimize the scalability of a BDCA system by automatically tuning Spark parameters*".

7.2.4.2 Reference inputs

Reference inputs specify the target to be achieved through adaptation [219]. For our approach, ideal scalability is the reference input. Ideal scalability implies that when the number of computing nodes is doubled, the job completion time (i.e., data processing time) is reduced by half. For example, if a BDCA system running on four nodes takes 40 sec to complete a job, increasing the number of nodes to eight should take 20 sec to complete the same job. According to Equation 7.1, the scalability score for ideal scalability is **1.0**, which *SCALER* aims to achieve.

7.2.4.3 Measured outputs

Whilst reference inputs are specified by a user, measured outputs are the actual measures collected from a running system [219]. The measured outputs are then compared with the reference inputs to assess to what extent the system has achieved the target state (indicated by reference input) through adaptation. For our adaptation approach, measured outputs are the scalability scores calculated when the system is under operation. The scalability score is then compared with the scalability score for the ideal scalability (i.e., 1.0) to assess the extent to which our adaptation approach has achieved its target.

7.2.4.4 Adaptation trigger

An adaptation trigger defines the condition for triggering the adaptation process [219]. In our approach, we define a threshold value for the scalability score. Our approach constantly monitors the scalability score of a system and whenever the scalability score is less than the threshold value, the adaptation is triggered to optimize the scalability score. In Algorithm 7.1, line 4-6 specifies the adaptation trigger condition. In order to determine the threshold value for the evaluation of our approach, we first calculated the scalability score for the 36 use cases (i.e., 9 Parameter Configurations × 4 datasets) reported in Section 7.3.2 using Equation 7.1 to Equation 7.5. For calculating the scalability scores, we used the same value (i.e., 0.2) for all weights (ω_1 – ω_{10}) in Equation 7.1 to Equation 7.5 to specify the same level of importance for all the gaps between ideal and achieved response time. We

computed the mean scalability score of the 36 use cases, which gives a value of 0.5135. Although the value 0.5135 underlines the mean scalability score for the considered scenario that can be set as a threshold, we increased the threshold value by setting the value of *incrementthreshold* equal to 0.0365 (Algorithm 7.1). This is because setting *incrementthreshold* to a positive value makes the evaluation of *SCALER* more robust. Adding *incrementthreshold* to the mean scalability score gives us a value of 0.55, which is the final threshold value for triggering adaptation.

7.2.4.5 Control actions

Once adaptation is triggered, the control actions are automatically taken by our adaptation approach to adapt a system. The adaptation process is only triggered when a system's scalability score gets below the threshold scalability score and so the control actions aim to make the scalability score above the threshold. In our approach, the control actions execute a system with different Spark parameter configurations with the aim to find a combination of parameters with which a system's scalability score gets above the threshold scalability score. The control actions only try changing the parameters that have a significant impact on scalability as determined in RQ2 (Section 7.3.2). These parameters with a significant impact on scalability are presented in Table 7.3 with the default and modified value for each

Algorithm. 1. Algorithm for Adapting Configuration Setting of Spark-based BDCA System

Inputs: $S_{Threshold}$ = Threshold Scalability Score
$S_{Increment}$ = Increment Scalability Threshold
$CPV = \{CPV_1, CPV_2, \ldots, CPV_n\}$ // *Set of Combination of Parameter Values*
$S_{optimal}$ = The most optimal scalability score
R_t = The number of times each execution is repeated
Output: $CPV_{optimal}$ = CPV with scalability score above $S_{Threshold}$
Steps:

```
1.  CPV_optimal ← CPV_default        // CPV_default = {A, A, A, A, A, A, A, A}
2.  S_default ← 0, S_optimal ← 0, R_t ← 3, S_Threshold ← 0.58, S_Increment ← 0.036   // Initialize the variables
3.  runBDCA with CPV_default         // execute the BDCA system with default Spark configuration settings
4.  Calculate Scalability Score S_default of the BDCA system using Eq. 1
5.    if S_default > S_Threshold then
6.    |   CPV_optimal ← CPV_default
7.    |   return CPV_optimal
8.    end if
9.    for i = 1 to CPV.legnth do
10.   |   CPV [i] ← B          //change the value of parameter from A to B as mentioned in Table 5
11.   |   for j = 1 to R_t do
12.   |   |   runBDCA with 1 worker node
13.   |   |   runBDCA with 6 worker nodes
14.   |   |   runBDCA with 8 worker nodes
15.   |   end for
16.   |   Calculate Trend using Eq. 4   // To investigate scalability trend from 6 to 8 nodes
17.   |   if Trend < 0 then
18.   |   |   CPV [i] ← A
19.   |   |   continue
20.   |   end if
21.   |   for k = 1 to R_t do
22.   |   |   runBDCA with 2 worker nodes
23.   |   |   runBDCA with 4 worker nodes
24.   |   end for
25.   |   Calculate Scalability Score S[i] for CPV [i] using Eq. 1
26.   |   if S[i] > S_optimal then
27.   |   |   S_optimal ← S[i]
28.   |   |   CPV_optimal ← CPV[i]
29.   |   |   if S[i] > S_Threshold then
30.   |   |   |   break
31.   |   |   end if
32.   |   end if
33.   end for
34. return CPV_optimal
```

Table 7. 3 Spark parameters used in the adaptation and their potential value options

ID	Spark Parameters	Default Value	Modified Value
P1	Spark.executor.memory	1024 (P1-A)	1250 (P1-B)
P2	Spark.shuffle.sort.bypassMergeThreshold	200 (P2-A)	400 (P2-B)
P3	Spark.shuffle.compress	TRUE (P3-A)	FALSE (P3-B)
P4	Spark.memory.storageFraction	0.5 (P4-A)	0.7 (P4-B)
P5	Spark.shuffle.file.buffer	32k (P5-A)	64k (P5-B)
P6	Spark.reducer.maxSizeInFlight	48m ((P6-A)	96m (P6-B)
P7	Spark.memory.fraction	0.6 (P7-A)	0.8 (P7-B)
P8	Spark.serializer.objectStreamReset	100 (P8-A)	-1 (P8-B)
P9	Spark.rdd.compress	FALSE (P9-A)	TRUE (P9-B)

Table 7. 4 Combination of Parameter Values (CPV) executed for identifying optimal CPV

CPV ID	Combination of Parameter Values (CPV)	P1	P2	P3	P4	P5	P6	P7	P8	P9
1	{A, A, A, A, A, A, A, A, A}	A	A	A	A	A	A	A	A	A
2	{B, A, A, A, A, A, A, A, A}	B	A	A	A	A	A	A	A	A
3	{A, B, A, A, A, A, A, A, A}	A	B	A	A	A	A	A	A	A
512	{B, B, B, B, B, B, B, B, B}	B	B	B	B	B	B	B	B	B

parameter. The modified values presented in Table 7.3 are chosen based on academic and industrial recommendations [248, 249]. For example, the execution memory can be set as 1024m (default value) or 1250m (modified value). In Table 7.4, 'A' represents the default value and 'B' represents the modified value for a parameter. For instance, P1-A and P1-B are default and modified values for parameter P1 (*Spark.Executor.Memory*). Some sample Combinations of Parameter Values (CPV) are shown in Table 7.4.

Since our approach considers a total of nine parameters each with two possible values, there are a total of 512 (2^9) CPVs. The execution of these many CPVs to find the CPV with a scalability score that is above the threshold is a computationally expensive task. The time required to execute and search through the large search space of 512 potential CPVs will outweigh the gain expected through adaptation. It is also worth noting that state-of-the-art tuning approaches (e.g., [250, 251]) do follow the strategy of searching through the entire search space. However, it is computationally feasible for these approaches as these approaches aim to tune for optimizing response time. Calculating response time requires the system to execute only once but for calculating scalability score as required for our approach, the system needs to be executed at least five times with a different number of computing nodes. We, therefore, employed the following optimization techniques to reduce the computational time with minimal impact on the accuracy of our approach.

Eliminating CPVs with negative trend: Before calculating the scalability score of a CPV for an entire curve obtained through executing the system with 1, 2, 4, 6, and 8 nodes, we calculate the scalability trend (Equation 7.4) from six to eight nodes. If the trend is negative, indicating that the response time is increased as we increase the cluster size from six to eight nodes, it is deducted that the CPV is not a candidate for the potential CPV with scalability score above the threshold. The reason we include merely the transition from 6 to 8 nodes in Equation 7.4 is the time overhead. To illustrate the impact of including more transitions in Equation 7.4 on the time to calculate Trend, we take two sample cases - (a) using the only transition from 6 to 8 nodes and (b) including two transitions i.e., from 4 to 6 nodes and from 6 to 8 node in Equation 7.4. We then apply the two sample cases on the hypothetical scenarios presented in Figure 7.2. On average, the time required to calculate Trend in case (a) is 9.24 sec while

the average time to calculate the trend in case (b) is 14.9 sec. Hence, the time required to calculate the trend in case (a) is 38.12% less than the time required to calculate the Trend in case (b). This difference in time required to calculate Trend increases with an increase in including more transitions (e.g., from 1 to 2 and 2 to 4 nodes) in Equation 7.4. *Keeping change of parameter value with a positive impact:* If changing value for a parameter improves the scalability score, the changed value is kept the same for the next CPV. For example, CPV_2 achieves a better scalability score than CPV_1 by changing the value of *spark.executor.memory* from 1024 to 1250. However, the scalability score of CPV_2 is not above the threshold value, our algorithm will not select CPV_2 rather will execute CPV_3 but with the *spark.executor.memory* value of 1250 as it already showed a better scalability score.

The adaptation algorithm employed in *SCALER* is presented as Algorithm 7.1. If the scalability score of the BDCA system with default parameter settings is below the threshold (line 4-6), adaptation process is triggered. The first parameter in the default CPV is changed from its default value and then the *Trend* is calculated using Equation 7.4 to investigate the trend of scalability from six nodes to eight nodes. If the *Trend* is negative, the parameter value is changed to its default value (line 15-18). On the other hand, if the *Trend* is positive, the BDCA system is executed with two and four worker nodes to get the entire scalability curve (line 19-22). After getting the scalability curve, the scalability score is calculated for the CPV. If the scalability score is higher than the previous best scalability score, the optimal CPV is updated. Finally, the scalability score of the CPV is compared with the threshold scalability score (line 27-29). If the scalability score is above the threshold, the CPV is selected for the future operation of the system.

The variable R_t in Algorithm 7.1 specifies the number of times each execution is repeated. Such a repetition of execution is required to remove (any) experimental fluctuations. We set R_t equal to three – indicating to repeat each execution three times. We then take the mean of the response time determined in the three executions for subsequent calculation of scalability score. It is important to note that Algorithm 7.1 only restricts adaptation trigger based on predefined threshold i.e., adaptation is triggered only if the scalability score is less than the predefined threshold. Algorithm 7.1 does ensure that it will return a CPV with scalability score either equal or better than the previously running CPV. Algorithm 7.1 does not guarantee that it will always return a CPV with scalability score above the predefined threshold, however, we did not observe any such case based on our results presented in Section 7.3.3.

7.3 Results

In this section, we present our results in the form of answers to the three research questions presented in Section 7.1.

7.3.1 Scalability of BDCA system with default Spark configuration settings

This section answers the RQ1 i.e., *How a BDCA system scales with default Spark configuration settings?* This research question investigates the very premise of the work reported in this chapter i.e., to confirm whether or not a BDCA system scales ideally. Ideal scalability implies that the BDCA system makes full use of the additional resource provided by scaling [252]. For instance, when the number of worker nodes is doubled, the response time (e.g., training time and prediction time) of the system should be reduced to half. If the BDCA system scales ideally, there would be no value added by our work reported in this chapter.

Figure 7.3 shows the scalability curves with default Spark settings for training time and prediction time/testing time of the four datasets. The dotted line in Figure 7.3 denotes ideal scalability and the solid line denotes achieved scalability. In the training phase, a system scales almost ideally as the number of worker nodes increases from one to two. The impact of adding further nodes until six nodes is negligible. After six nodes, the addition of nodes has a negative impact on scalability, i.e., the training time slightly increases. We used Equation 7.1 to quantify the ideal and achieved scalabilities. The ideal scalability score for each dataset is 1. The scalability scores for achieved scalabilities are; KDD – (-0.14), DARPA – 0.53, CIDDS – 0.68, and CIDIDS2017 – 0.61. This trend is largely in line with the number of instances in each dataset. For instance, CIDDS having the largest number of instances achieves the best scalability and KDD with the smallest number of instances achieves the lowest scalability. The deviation from ideal scalability for each dataset is calculated as: deviation = (1- scalability score) *100. The deviation from ideal scalability for each dataset is found to be; KDD – 114%, DARPA – 47%, CIDDS – 32%, and CIDDS2017 – 39%. On average, the achieved scalability of a BDCA system deviates from the ideal scalability by 58%. The scalability with respect to prediction time is abrupt. This is because of the very quick response of the system (i.e., in milliseconds) during the testing phase. Such abrupt changes in prediction time make our findings unreliable to be used for scalability analysis. Therefore, in the rest of this chapter, following the approach used in the related studies BDCA studies (e.g., [153]), we only report our findings with respect to the training time.

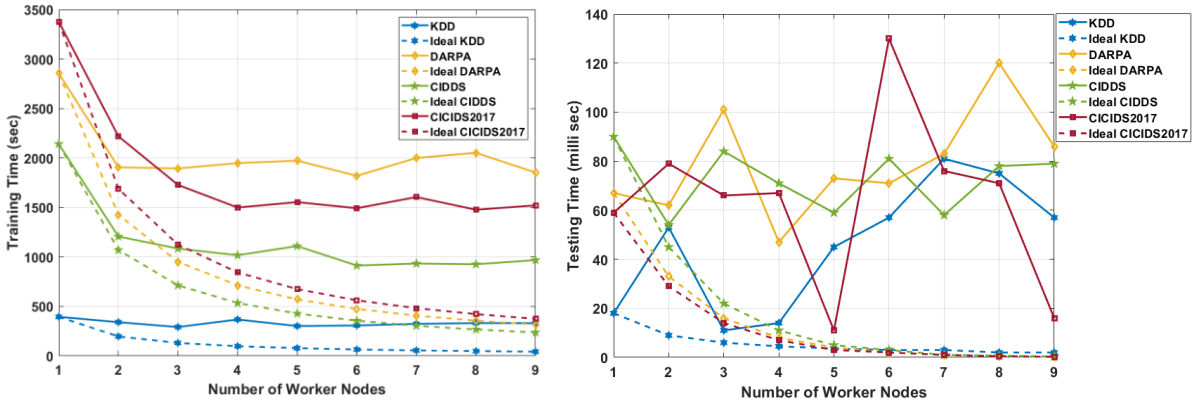


Figure 7. 3 Ideal and achieved scalability with default Spark settings during training phase and testing phase for KDD, DARPA, CIDDS, and CICIDS2017 datasets

The summary answer to RQ1: *A BDCA system with default Spark configuration settings does not scale ideally. The average deviation from ideal scalability is 58%.*

7.3.2 Impact of Spark configuration parameters on scalability of a BDCA system

This section answers RQ2 i.e., *What is the impact of tuning Spark Configuration Parameters on the scalability of a BDCA system?* Table 7.5 shows the default values and the modified values for the 11 configuration parameters (described in Section 7.2.4) and the scalability scores achieved with the default and modified settings. Figure 7.4 to Figure 7.7 show the difference in scalability graphs with the default and modified values for each of the 11 parameters for KDD, DARPA, CIDDS, and CICIDS2017 dataset. For instance, as shown in Table 7.5, changing the value of *spark.rdd.compress* from FALSE (default) to TRUE changes the scalability score of the system from 0.14 to 0.92 for KDD, 0.53 to 0.38 for DARPA, 0.68 to 0.69 for CIDDS, and 0.61 to 0.64 for CICIDS2017. The same trend continues for the first nine

parameters shown in Table 7.5, where modifying the value of the parameters leads to significant change in the scalability score. The last two parameters (i.e., *spark.driver.memory* and *spark.shuffle.memoryFraction*) do not significantly impact the scalability. For example, as presented in Table 7.5, changing the value of *spark.shuffle.memoryFraction* from '/' (default) to 0.4 brings an insignificant change in scalability score for KDD (0.14 to 0.13), DARPA (0.53 to 0.54), CIDDS (0.68 to 0.54), and CICIDS2017 (0.61 to 0.6).

We assess whether modifying a parameter value has a positive or negative impact on scalability. Table 7.5 shows that the positive or negative impact of modifying a parameter value varies from one dataset to another. The underlined values in Table 7.5 indicate the negative impact of changing the default value for the parameter. For example, changing the value of *spark.shuffle.compress* from TRUE to FALSE has a positive impact on scalability with KDD, DARPA, and CIDDS but a negative impact on scalability with CICIDS2017 i.e., scalability score has decreased. Similarly, the default value (100) of *spark.serializer.objectStreamReset* achieves better scalability for KDD and CICIDS2017 while the modified value (-1) achieves better scalability for DARPA and CICIDS2017. This finding underlines a correlation between the dataset and Spark configuration parameters. Hence, it can be asserted that the Spark configuration parameters need to be configured as per the type of the dataset. In other words, this finding invalidates the reuse of a single Spark configuration setting across multiple datasets.

Table 7.6 presents the ranking of the studied Spark parameters based on their impact on scalability. The impact is calculated as the difference between the scalability score with default Spark setting and

Table 7. 5 Scalability score with default and modified value for the 11 studied parameters

ID	Spark Parameters	Configuration		Scalability Score			
		Default Value	Modified Value	KDD	DARPA	CIDDS	CICIDS2017
	Default Settings			-0.14	0.53	0.68	0.61
P1	Spark.executor.memory	1024	1250	0.31	0.62	0.72	0.61
P2	Spark.shuffle.sort.bypassMergeThreshold	200	400	*0*	0.81	0.68	0.7
P3	Spark.shuffle.compress	TRUE	FALSE	0.36	0.64	0.82	*0.49*
P4	Spark.memory.storageFraction	0.5	0.7	0.5	0.67	*0.51*	*0.46*
P5	Spark.shuffle.file.buffer	32k	64k	0.15	*0.38*	0.68	*0.46*
P6	Spark.reducer.maxSizeInFlight	48m	96m	0.68	0.68	*0.52*	*0.59*
P7	Spark.memory.fraction	0.6	0.8	*0.01*	*0.44*	*0.59*	0.61
P8	Spark.serializer.objectStreamReset	100	-1	0.71	*0.43*	*0.44*	0.67
P9	Spark.rdd.compress	FALSE	TRUE	0.92	*0.38*	0.69	0.64
P10	Spark.shuffle.memoryFraction	\(Deprecated)	0.4	*-0.13*	0.54	*0.54*	0.62
P11	Spark.driver.memory	1024	1600	-0.15	0.38	*0.53*	*0.60*

Table 7. 6 Ranking of the studied parameters based on their impact on scalability

ID	Spark Parameters	Overall Ranking	Ranking with Respect to Datasets			
			KDD	DARPA	CIDDS	CICIDS2017
P1	Spark.executor.memory	7 (0.58)	6 (0.45)	10 (0.08)	8 (0.04)	10 (0.0)
P2	Spark.shuffle.sort.bypassMergeThreshold	8 (0.51)	2 (0.14)	1 (0.28)	10 (0.0)	4 (0.09)
P3	Spark.shuffle.compress	5 (0.87)	5 (0.50)	7 (0.11)	6 (0.144)	3 (0.12)
P4	Spark.memory.storageFraction	4 (1.11)	4 (0.64)	6 (0.14)	2 (0.17)	1 (0.158)
P5	Spark.shuffle.file.buffer	6 (0.59)	7 (0.30)	2 (0.156)	11 (0.0)	2 (0.156)
P6	Spark.reducer.maxSizeInFlight	3 (1.15)	3 (0.82)	3 (0.154)	3 (0.16)	7 (0.02)
P7	Spark.memory.fraction	9 (0.33)	9 (0.15)	9 (0.09)	7 (0.09)	11 (0.0)
P8	Spark.serializer.objectStreamReset	1 (1.26)	2 (0.85)	8 (0.10)	1 (0.24)	5 (0.06)
P9	Spark.rdd.compress	2 (1.25)	1 (1.06)	4 (0.153)	9 (0.01)	6 (0.03)
P10	Spark.shuffle.memoryFraction	11 (0.17)	11 (0.01)	11 (0.01)	5 (0.146)	8 (0.01)
P11	Spark.driver.memory	10 (0.32)	8 (0.29)	5 (0.151)	4 (0.15)	9 (0.01)

the modified Spark setting. The number in brackets in Table 7.6 specifies the difference in scalability score between the default settings and the modified settings. Such a ranking is useful in prioritizing the tuning of particular parameters, i.e., parameters with a significant impact. Overall, our findings show that with respect to scalability, *Spark.serializer.objectStreamReset* is the most impactful and *Spark.shuffle.memoryFraction* is the least impactful Spark parameter. It is worth noting that *Spark.shuffle.compress* significantly impacts the training time as can be observed from Figure 7.4 – Figure 7.7. However, a significant impact on training time does not necessarily mean a significant impact on scalability. That is why it is not ranked as the most impactful with respect to scalability. Table 7.6 also depicts that the ranking of parameters varies with respect to datasets. For example, *Spark.shuffle.sort.bypassMergeTreshold* is ranked as 1st and 2nd for DARPA and KDD datasets respectively, however, it is ranked as 10th for CIDDS dataset. This further underlines the correlation between the parameter setting and the underlying dataset. As stated earlier and illustrated by the ranking, the two parameters (i.e., *spark.driver.memory* and *spark.shuffle.memoryFraction*) are ranked at the bottom due to their insignificant or minor impact on the scalability score.

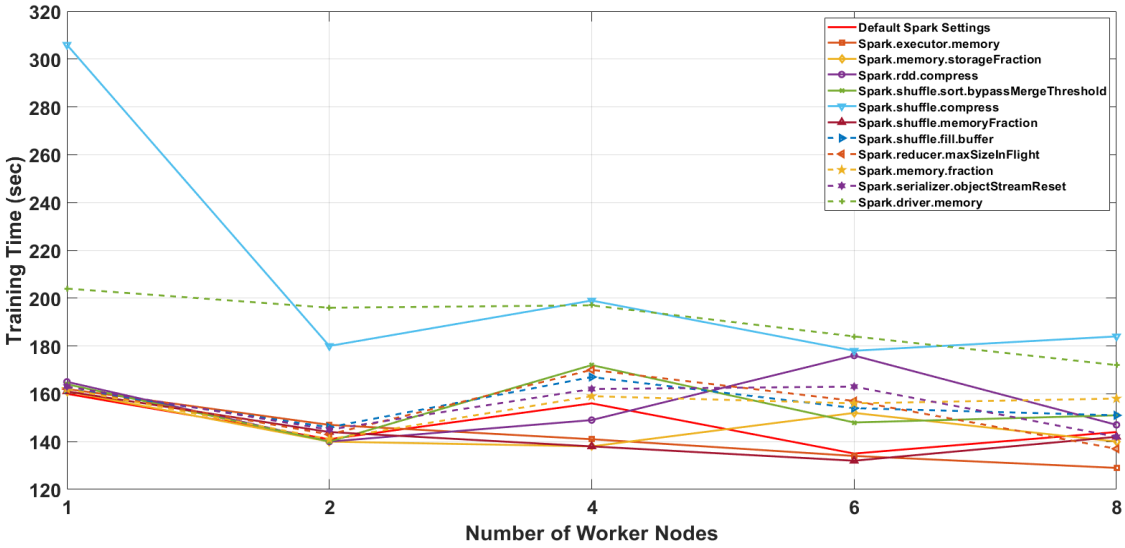


Figure 7. 4 Impact of modifying the value of parameters on the scalability with KDD dataset

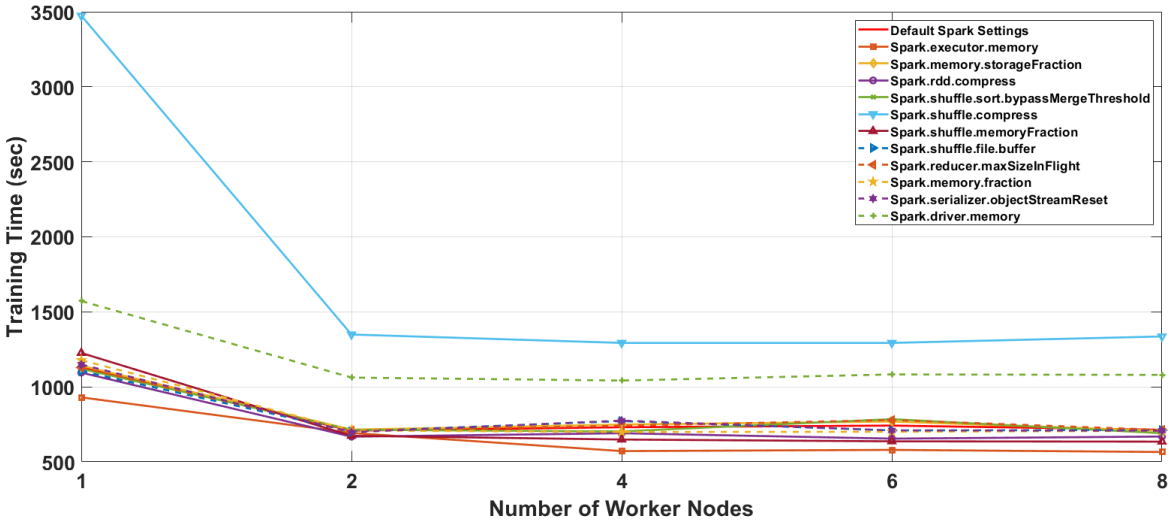


Figure 7. 5 Impact of modifying the value of parameters on the scalability with DARPA dataset

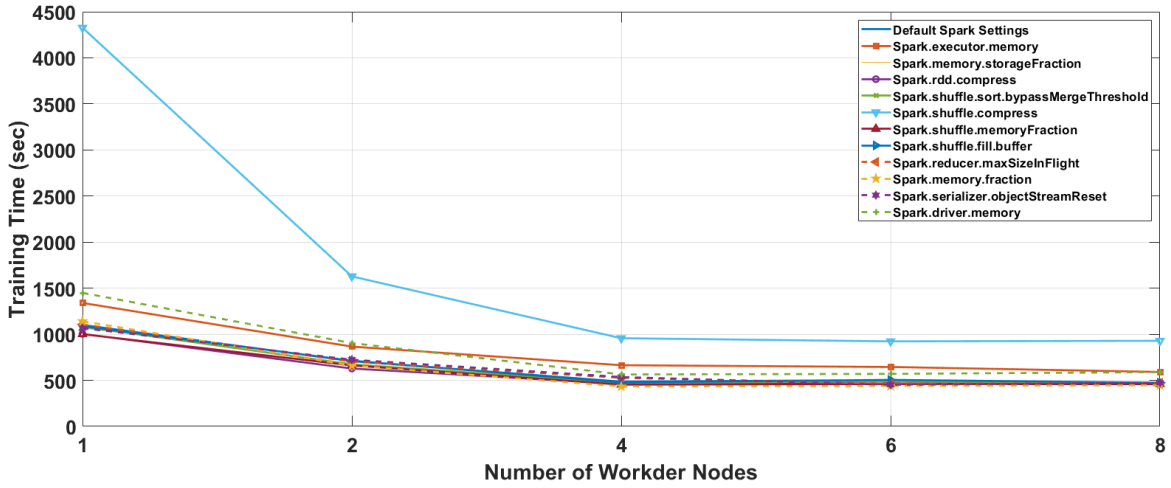


Figure 7. 6 Impact of modifying the value of parameters on the scalability with CIDDS dataset

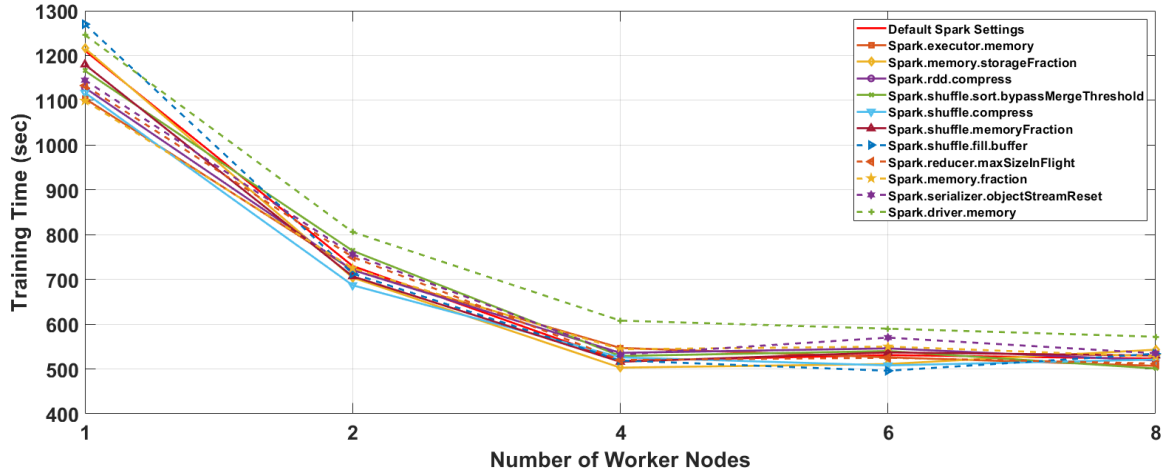


Figure 7. 7 Impact of modifying the value of parameters on the scalability with CIDIDS2017 dataset

The summary answer to RQ2: *Modifying the default value of 9 out of 11 studied Spark parameters significantly impact the scalability of a BDCA system. Each security dataset requires a separate configuration of Spark parameters for achieving optimal scalability. With respect to scalability, Spark.serializer.objectStreamReset is the most impactful and Spark.shuffle.memoryFraction is the least*

7.3.3 Optimizing scalability of a BDCA system

This section answers RQ3 i.e., *How to improve the scalability of a BDCA system?* We have proposed a parameter-driven adaptation approach, *SCALER*, for improving the scalability of a BDCA system. The adaptation approach has already been described in Section 7.2.4. Here, we evaluate the effectiveness of our approach with respect to the following two research questions.

7.3.3.1 RQ3.1: Does *SCALER* improve the scalability of a BDCA system?

We assess the scalability improvement by comparing the scalability score achieved by our system exactly before and after adaptation. In order to realize adaptation, we experimented with two scenarios i.e., baseline and change in input data. In the *baseline* scenario, the BDCA system is processing a particular dataset such as KDD. In the *change in input data* scenario, the input to the system is changed from one dataset to another e.g., from KDD to CIDDS. Upon the change in the dataset, *SCALER*

calculates the scalability score for the new dataset (i.e., CIDDS) and if the scalability score is lower than the predefined threshold (0.55), adaptation process is triggered. Table 7.7 shows the scalability scores before and after adaptation for the four datasets. On average, *SCALER* improves scalability by 16%. With respect to datasets, the highest improvement is achieved for CIDDS (22% improvement) followed by KDD (19% improvement), and CICIDS2017 (7%). Since the scalability score of DARPA is higher than the threshold score of 0.55, adaptation is not triggered for DARPA dataset. The trend with respect to dataset correlates with the number of features in each dataset – the smaller the number of features, the larger is the improvement. The scalability score for CIDDS having the smallest number of features (i.e., 12) improves the most while the CICIDS2017 with the largest number of features (i.e., 80) benefits the least from the adaptation.

The summary answer to RQ3.1: *SCALER improves the scalability of the BDCA system by 16%. The smaller the number of features in the dataset, the larger is the benefit, in terms of scalability improvement, from SCALER*

Table 7. 7 Scalability score of our BDCA system before and after adaptation

Dataset	Scalability Score Before Adaptation	Scalability Score After Adaptation	Improvement (%)
KDD	0.44	0.63	19 %
DARPA	0.70	0.70	0 %
CIDDS	0.51	0.72	22 %
CICIDS2017	0.53	0.60	7 %

7.3.3.2 RQ3.2: How long does it take for SCALER to adapt a BDCA system for optimal scalability?

The adaptation time underlines the speed with which *SCALER* adapts the system. The adaptation time is calculated as the time between the point of time adaptation is triggered to the point when the system gains a stable state i.e., the adaptation process is terminated [219]. Figure 7.8 shows the adaptation time of *SCALER* for the four datasets. On average, it takes around 435 minutes for *SCALER* to adapt a system, i.e., to bring a system to a state where the scalability of the system is above the predefined threshold. It is worth noting that unlike previous studies (e.g., [226, 250]) that adapt for improving the response time, our approach takes more time due to the generation of scalability curve instead of a single point required for response time optimization. The adaptation time is mainly elapsed in executing the system with different CPVs (Table 7.4) to identify the CPV with which the system has a scalability score above the threshold. With respect to datasets, the adaptation time is longest (i.e., 951.4 min) for CICIDS2017 followed by CIDDS (266.2 min), and KDD (87.4 min). This trend is in accordance

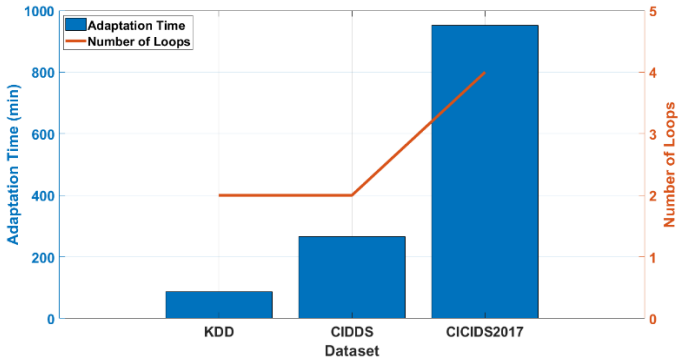


Figure 7. 8 Time taken by *SCALER* to adapt a BDCA for optimal scalability

with the size of each dataset. For example, the system takes the longest time to adapt itself for CICIDS2017 that is largest in size and takes the shortest time to adapt itself for KDD dataset, which is the smallest in size. In Figure 7.8, the number of loops indicates the number of CPVs executed at runtime by SCALER to converge towards a CPV with which the system has a scalability score above the threshold. The larger the number of CPVs executed for convergence, the larger is the adaptation time and vice versa. For KDD and CIDDS, it took two loops while for CICIDS2017 it took four loops for our approach to identify the CPV with scalability score above the threshold.

It is worth noting that the adaptation time is larger than the actual job completion time (i.e., training time). For example, mean training time for our BDCA system with CIDDS dataset is 19.4 min while the adaptation time for our BDCA system with CIDDS dataset is 266.2 min. This is because in order to determine training time, the system needs to be executed only once. However, to determine scalability score, the system needs to be executed multiple times with different number of nodes (i.e., 1, 2, 4, 6, and 8 nodes in our case). Adaptation time is elapsed in determining scalability score for different parameter combinations; therefore, adaptation time is larger than training time. Similar to most of the tuning approaches (e.g., [38, 41, 60]), the real advantage of SCALER is in the execution of recurring jobs, which is a common phenomenon and equally applicable to security analytics [61, 62]. Some recent studies [61, 62] reveal that around 40% of data analytics jobs are recurrent jobs. The current job is executed for the sake of tuning; therefore, it does not benefit from the tuning. However, the recurring and/or subsequent jobs benefit from the already tuned system. For example, SCALER improves the scalability score of a job (i.e., training the BDCA system with CIDDS dataset) from 0.51 to 0.72 – an improvement of around 22%, which in turn translates into a reduction of training time from 15.4 min to 11.2 min with an eight node cluster. Now, since the system is tuned, when the system executes the same or similar job, it will take 11.2 min to complete the job instead of 15.4 min.

The summary answer to RQ3.2: *SCALER takes around 435 min to adapt a BDCA system to a dataset. The time taken to adapt a system is directly proportional to the size of the dataset.*

7.4 Discussion

In this section, we discuss the results presented in the previous section from two perspectives i.e., benefits to the security operator and performance bottlenecks.

Benefits to security operator: We now turn to the question that how the findings of the work reported in this chapter are useful for operator of a BDCA system. In practice, BDCA systems are deployed and operated using default Spark settings [239]. The first take away from our findings is the realization that, at least from the scalability point of view, default Spark settings are not optimal. Furthermore, our findings indicate that on average the deviation from ideal scalability is around 53%. The finding motivates a security operator to assess the scalability of the BDCA system after it is deployed. If the system scales poorly, the security operator can manually tune the parameters to improve system's scalability. The identification of impactful parameters in Section 7.3.2 further facilitates the security operator to only assess the tuning of parameters that has a significant impact on system's scalability. *SCALER* saves the security operator's time and effort that is to be invested in finding the right combination of parameters with which the system can scale in an optimal way.

Experimental bottlenecks: During our experimentation, we faced several performance bottlenecks related to the hardware resources and Spark processing framework. We discuss those issues for the benefit of other researchers, who may come across the same issues. Our experiments produced

temporary data during job execution on Spark, which consumes a lot of disk space on worker nodes. Since each worker node has a limited disk space (10 GB in our case), the temporary data can exceed the disk limit of the worker node. In such a case, the worker node becomes unhealthy and so the master node does not assign the worker node any further tasks. To deal with this issue, we delete the temporary data produced on worker nodes. However, we ensured that critical data such as HDFS block files and block meta files are not deleted. Debugging becomes a serious concern in distributed data processing. We also initially faced challenges with debugging the failures (e.g., node failure) during our experiments. For instance, running a data processing job with eight worker nodes, where two worker nodes are already unhealthy, consumes the amount of time but produce useless results. This is because the experiment was designed for eight nodes, but two nodes were in unhealthy state and we were not aware of it. To handle this issue, we designated a path through the variable *yarn.nodemanager.log-dirs* as the path for saving operational logs. We used to constantly check the logs to identify any issues before running the experiment.

7.5 Threats to Validity

External Validity: In this chapter, we investigated a specific BDCA system that is using a particular ML algorithm (Naïve Bayes) and big data framework (Spark). Therefore, our findings may not generalize to all kinds of BDCA systems. However, it is important to note that the aim of this study is not to show results that generalize to all BDCA systems but rather to show that parameter configuration of the underlying big data framework impacts BDCA system's scalability. Nonetheless, future studies aimed at generalizing our findings will be fruitful. ***Construct Validity:*** The number of value options (i.e., two) we investigated for some parameters limit exploration of the parameter space. Since the modification of a parameter value (e.g., from 1024 to 1250) from its default value shows a significant impact on scalability, investigating other modifications (e.g., 1024 to 2056) from the default value can only strengthen our findings but cannot contradict them. Our study only investigated a limited number of parameters for their impact on scalability. Even if other Spark parameters do not impact scalability, our findings for the studied parameters still remain valid. The experimental bottlenecks presented in Section 7.4 also pose a threat to the correctness of the measurements recorded during the experiments. Whilst it is quite challenging to completely overcome such bottlenecks, we ensured to reduce their impact as much as possible as explained in Section 7.4. ***Internal Validity:*** *SCALER* takes around 435 minutes to select Spark configuration with a scalability score above the threshold. The limitation about large adaptation time can be mitigated in future by (i) reducing the number of parameters considered during tuning through techniques such as Lasso linear regression and (ii) similar to [224] and [253], using representative datasets of smaller size instead of using the original datasets.

7.6 Related Work

In this section, we position the novelty the work reported in this chapter with respect to other related studies.

Among the 74 papers reviewed in Chapter 2, 40 papers emphasized the importance of scalability for a BDCA system. However, only a few studies (i.e., [35, 153, 206, 254, 255]) empirically investigated the scalability of a BDCA system. Lee et al. [35] investigated the scalability of a Hadoop-based BDCA system on a 30-node cluster and observed that the execution time improves in proportion to the hardware resource from 5 to 30 nodes. Du et al. [254] studied the scalability of a Storm-based BDCA

system on a five-node cluster and observed that the system failed to achieve ideal scalability due to extra task scheduling and communications overhead between *spout* and *bold* phases of Storm execution environment. Aljarah et al. [153] also studied the scalability of a Hadoop-based BDCA system on an 18 node cluster and found that the system scales abruptly as the number of nodes in the cluster are increased. For example, ideal speedup is observed from two to four nodes and 14 to 16 nodes while non-ideal speedup is observed for the rest of the scalability curve. The non-ideal speedup is attributed to the start-up of MapReduce jobs and storing intermediate results in HDFS. Las-Casas et al., [255] compared the scalability of two BDCA systems – one based on Hadoop and another based on Spark. The authors of [255] found that Spark scales better than Hadoop due to efficient use of caching in Spark. Xiang et al. [206] also explored the scalability of a BDCA system on a 30-node cluster and found that the response time decreases, although not ideally, with increase in number of nodes up till 25 nodes. After 25 nodes, the response time is increased, which the authors attribute to the excessive communication among nodes and disk read/write operation during MapReduce tasks. Although the previous studies have investigated the scalability of a BDCA system, none of the studies have quantified the scalability nor calculated its deviation from the ideal scalability. Furthermore, previous studies have only investigated the scalability with default settings. Ours is the first effort that (i) quantified the scalability with respect to four datasets (ii) assessed the deviation from the ideal scalability and most importantly (iii) investigated the impact of Spark configuration parameters on the scalability of a BDCA system.

A number of studies (e.g., [256-259]) have proposed methods for optimizing scalability of software systems in different domains. For example, Kyong et al. [256] proposed a docker contained based architecture for Spark-based scale up server, where the original scale up server is partitioned into several small server to reduce memory access overheads. Wu et al. [257] proposed a scalability improvement technique that learns from the interaction patterns among the services of a service-based application and accordingly adopt an optimized task assignment strategy to reduce the communication bandwidth and improve the scalability. Senger et al. [258] defined a scalability measure called input file affinity, which quantifies the level of file sharing among tasks belonging to a bag-of-tasks application (e.g., data mining algorithm). The paper (i.e., [258]) then proposed scalability improvement method that leverages the input file affinity measure to increase the degree of file sharing among the tasks. Canali et al., [259] presented a scalability improvement approach for cloud-based systems, which leverages the resource usage patterns (e.g., CPU, storage, and network) among the virtual machines in the cloud and accordingly group the virtual machines. It is important to note that some studies (e.g., [236-239]) have proposed tuning techniques for Spark-based systems with the objective to reduce response time. Such studies are largely impertinent to ours as they are focussed on response time and our study focusses on scalability, which are two very different quality attributes of a software system and are treated differently in the state-of-the-art [260]. Therefore, the approaches presented in [236-239] are not applicable for improving scalability. In general, the previous studies are largely orthogonal to our study. This is because (i) our study is the first of its kind that aims to improve scalability of Spark-based BDCA systems and (ii) employ a parameter-driven adaptation approach for improving the scalability of a BDCA system.

Parameter-driven adaptation is one of the commonly used adaptation approaches. Several studies (e.g., [214, 226, 261, 262]) have explored modifying the values of certain system's parameters to achieve various objectives such as high accuracy and improved security. Calinescu et al. [214] used KAMI model based on Bayesian estimator to modify model parameters at runtime for achieving reliability and quick response in a service-based medical assistance system. In [261], the authors argue that

parameters of software abstraction models such as Discrete Time Markov Chains (DTMC) should be constantly updated to achieve better accuracy. Accordingly, the authors of [261] proposed an adaptation method that leverage real time operational data of the system to keep the parameters up to date. Similarly, parameter-based adaptation is quite common in ML domain for adjusting model parameters. For example, Tongchim et al. [262] proposed a parameter-driven adaptation approach for adjusting the control parameters of genetic algorithms to achieve optimal accuracy. The approach in [262] divides the parameter space into sub-spaces and each sub-space evolves on separate computing nodes in parallel. As another example, Jiang et al. [226] proposed a parameter-driven adaptation approach that uses the temporal and spatial correlations among characteristics (such as size and velocity of objects) for finding the best set of configuration parameters for a convolutional neural network employed in a video analytics system. The adaptation approach proposed in [226] aims to balance resource consumption and system's accuracy. From adaptation point of view, our study differs from previous studies in two ways. First, ours is the first study to apply a parameter-driven adaptation approach in the domain of Spark based systems. Second, unlike our study, none of the previous studies have used parameter-driven adaptation for achieving improved scalability.

7.7 Chapter Summary

The exponential growth in the volume and velocity of security event data requires BDCA systems to be highly scalable. Therefore, in this chapter, we studied (i) how a Spark-based BDCA system scales with default Spark settings (ii) what is the impact of tuning Spark configuration parameters (e.g., execution memory) on the scalability of a BDCA system and (iii) proposed *SCALER* - a parameter-driven adaptation approach to optimize BDCA system's scalability. For this study, we developed an experimental infrastructure using a large-scale OpenStack cloud. We implemented a Spark-based BDCA system and used four security datasets to find out how the BDCA system scales, how Spark parameters impact scalability, and to evaluate our adaptation approach aimed at optimizing scalability. Based on our comprehensive experiments, we found that:

- With default Spark settings, a BDCA does not scale ideally. The deviation from ideal scalability is around 58%. The system scales better with large size datasets (e.g., CICIDS2017) as compared to small size datasets (e.g., KDD)
- 9 out of 11 studied Spark parameters significantly impact the scalability of a BDCA system. The impact of configuration parameters on scalability varies from one security dataset to another
- Our parameter-driven adaptation approach, *SCALER*, improves the mean scalability of a BDCA system by 16%.

From our findings, we conclude that practitioners should first tune the parameters of Spark before putting a Spark based BDCA system into operation. Such parameter tunning can significantly improve the scalability of the system. We also recommend that practitioners should not use someone else's pre-tuned parameter settings. The reason for this is that the best combination of Spark parameters varies from dataset to dataset. Our proposed adaptation approach is the first step towards facilitating practitioners to automatically tune Spark parameters for achieving optimal scalability. More generally, we assert that the field of big data analytics should pay attention to the impact of the configuration parameters of the big data frameworks on the various system qualities such as reliability, response time, and scalability.

Chapter 8

Conclusion and Future Work

Big Data Cyber Security Analytics (BDCA) systems are a new breed of software systems, which are increasingly used for cyber protection. BDCA systems are expected to possess several quality attributes among which response time, accuracy, and scalability are the most prominent. Given that BDCA systems are a new and complex class of software systems, there is a paucity of design knowledge for systematically designing BDCA systems that exhibit an optimal level of response time, accuracy, and scalability. In this thesis, we have aimed to propose and evaluate design knowledge and quality-centric approaches for designing BDCA systems. To this end, we have proposed, implemented, and evaluated design knowledge (e.g., architectural tactics and design guidelines) and a set of quality-centric design approaches for optimizing response time, accuracy, and scalability of BDCA systems. In the proposed approaches, we have focused on exploiting the optimization opportunities offered by the software architectural design and the underlying big data framework of a BDCA system. We have consistently built the proposed approaches based on the knowledge (e.g., architectural tactics) derived from the literature and the empirical findings (e.g., the impact of architectural tactics) obtained from the initial set of experiments. To evaluate the proposed approaches, we have taken an experimental approach by implementing the approaches on a large-scale cluster using several state-of-the-art datasets. The rich set of experiments, we have performed, can provide valuable insight into the design of BDCA systems and the optimization achieved in response time, accuracy, and scalability by our proposed approaches. In this chapter, we first summarize the key findings that offer answers to the research questions outlined in Chapter 1. We then suggest areas for future research.

8.1 Findings and Contributions

In the following, we summarize the key findings of this thesis with respect to the research questions presented in Chapter 1.

1. Through a systematic literature review reported in Chapter 2, we have (i) identified 12 most important quality attributes of a BDCA system and the motivation behind their importance in a BDCA system. These 12 quality attributes are response time, accuracy, scalability, reliability, usability, interoperability, adaptability, modifiability, generality, privacy assurance, security, and stealthiness (ii) identified and codified 17 architectural tactics used for achieving various quality attributes in a BDCA system. The tactics include six response time tactics, four accuracy tactics, two scalability tactics, three reliability tactics, and one security and usability tactic each (iii) revealed several research gaps and proposed recommendations for practitioners. The identification of the most important quality attributes and the catalogue of architectural tactics could serve as a useful design knowledge for practitioners to design BDCA systems.

2. With a comprehensive set of experiments in Chapter 3, we have shown the impact of various architectural tactics on the accuracy and response time of a BDCA system. We have found that contextual factors (i.e., execution mode, machine learning model, and data quality) play a vital role in enabling or disabling the architectural tactics to achieve accuracy and response time. Based on our experimental findings, we have proposed design guidelines that could guide the practitioners on how to optimally use the architectural tactics to the maximum of their ability with respect to the contextual factors.

3. In Chapter 4, we have (i) disclosed that the choice of architectural configuration (i.e., the set of components comprising the BDCA system) significantly impacts the accuracy and prediction time of a BDCA system (ii) proposed guidance models for guiding the design process (e.g., quality implication, design dependencies, and design constraints) of BDCA systems and (iii) shown through rigorous experimentation that our heuristic-based design approach (*TACTics*) is quite effective in selecting an optimal architectural configuration for BDCA systems at the design time. The use of *TACTics* optimizes accuracy by 4.66% and prediction time by 20.56%. Our findings highlight the importance of architectural configuration for BDCA systems. It is also important to note that *TACTics* does not attempt to replace an architect rather aims to enable the architect to consider a variety of factors (e.g., multiple quality attributes and dependencies) in decision making and end up with a BDCA design that is optimal, traceable, and justifiable.

4. We have found that the effectiveness of components (e.g., feature selection) of a BDCA system is sensitive to the quality and velocity of security event data. The selection of components directly impacts the accuracy and prediction time. Our architecture-driven adaptation approach, *ADABTics*, presented in Chapter 5 enables a BDCA system to operationalize and de-operationalize components at runtime according to the changes in the security event data. Through comprehensive experiments, we have shown that *ADABTics* optimizes accuracy and prediction time by 6.06% and 23.71% respectively with an average adaptation time of 432 sec. Our findings underline that (i) a static BDCA system (i.e., a BDCA system consisting of a fixed set of components) degrades its accuracy and prediction time with the changes in the security event data and (ii) employing an adaptation approach that adapts the set components of a BDCA system according to the changes in the security event data significantly optimizes accuracy and prediction time of the system.

5. An operating environment that experiences frequent changes in the security event data will challenge the effectiveness of *ADABTics* due to its high adaptation time. Therefore, we proposed a more agile adaptation approach (*QuickAdapt*) in Chapter 6. Our experimental results have shown that the adaptation time of *QuickAdapt* is 107 times lower than *ADABTics*. However, *QuickAdapt* falls short in reaching the optimization level achieved with *ADABTics* i.e., optimization of 4.51% in accuracy and 17.6% in prediction time with *QuickAdapt*. Hence, we can conclude that *ADABTics* should be employed in operating environments subjected to a lower frequency of changes while *QuickAdapt* should be employed in environments where the quality of security event data changes more often.

6. We have found that (i) a BDCA system with the default setting of Spark configuration parameters does not scale ideally as demonstrated in Chapter 7. The average deviation from

ideal scalability is found to be 58% (ii) changing the values of the configuration parameters of Apache Spark significantly impacts the scalability of the Spark-based BDCA system (ii) our parameter-driven adaptation approach (*SCALER*), which automatically tunes the parameters to optimal values, optimizes the scalability of a BDCA system by 16%. Our findings highlight the importance of exploring and exploiting the configuration parameter space of the underlying big data framework (i.e., Apache Spark) for optimizing the scalability of a BDCA system.

8.2 Future Directions

We believe this thesis makes a significant contribution towards the goal of designing a BDCA system and optimizing accuracy, response time, and scalability of a BDCA system. However, we also believe that this thesis has revealed several areas in the field of BDCA that can be explored in the future. These avenues for future research are discussed in the following.

8.2.1 Generalization with respect to big data frameworks, algorithms, and datasets

As discussed in Chapter 1, we used two big data frameworks, five machine learning algorithms, and four datasets in the research reported in this thesis. A natural extension of the reported research would be to evaluate our approaches (e.g., *ADABTics* and *QuickAdapt*) with other big data frameworks, machine learning algorithms, and security datasets. An extension along this line will provide a generalization of the findings reported in this thesis. Moreover, such an extension can reveal interesting comparisons (e.g., optimization potentials) among the frameworks. For the proposed extension, we highlight Apache Storm[14], Apache Samza[15], and Apache Flink[16] as the potential big data frameworks. Similarly, several security datasets [263] and machine learning algorithms [144] are available for the research direction.

8.2.2 Deep learning for big data cyber security analytics

In this thesis, we have used only traditional machine learning algorithms (e.g., Naïve Bayes and Random Forest) for anomaly-based attack detection as mentioned in Chapter 1. However, learning from training data with traditional machine learning algorithms is shallow and often requires feature engineering. With the increasing complexity of cyber-attacks, the traditional machine learning algorithms might lead to a decrease in the attack detection accuracy [264]. Hence, it would be interesting to investigate the utilization of deep learning approaches (e.g., recurrent neural networks) for anomaly-based attack detection and compare their accuracy, response time, and scalability with that of traditional machine learning algorithms as reported in this thesis. Whilst deep learning is leveraged in traditional cyber security systems (e.g., intrusion detection systems [264, 265]), it is not so far investigated for BDCA. One possible reason being the lack of libraries for distributed implementation of deep learning until the recent past. However, such libraries (e.g., DeepLearning4J [266]) are available now, hence, deep learning for BDCA should be investigated.

[14] https://storm.apache.org/
[15] http://samza.apache.org/
[16] https://flink.apache.org/

8.2.3 Extension to other domains

Whilst the various approaches (e.g., *ADABTics* and *SCALER*) presented in this thesis aim to optimize the accuracy, response time, and scalability of a BDCA system, the same approaches can be customized and extended to other domains. For instance, traffic optimization is one such domain where real-time and accurate analysis of the traffic data is required to determine the most optimal path to destination, time to destination, and any blockages due to accidents [267]. The architecture-driven adaptation approaches presented in Chapter 5 and Chapter 6 can be extended to adapt a traffic optimization system, which comprises another flavor of components and interactions as compared to a BDCA system, according to the changes in the on-ground traffic situation and associated traffic data. Similarly, banking is another domain where big data analytics is making its mark. In banking, big data analytics is used to analyze large volumes of data to understand in (near real-time) customer behavior, promote the right product, and increase revenue [268]. Similar to security data, the generation of banking data (e.g., transactions) increases and decreases with time [269]. For example, heavy load is observed during working hours and at the end of a month. Accordingly, the banking big data analytics systems need to scale as per the workload. Therefore, the adaptation approach presented in Chapter 7 can be applied to banking big data systems for optimizing the system's scalability.

8.2.4 Adaptation for feature selection technique and ML algorithm

The adaptation approaches presented in this thesis either adapt the architectural configuration (Chapter 5 and Chapter 6) or the parameter configuration of the underlying big data framework (Chapter 7) of a BDCA system. However, the internal implementation of various components (e.g., feature selection and anomaly-based attack detection) remains the same. For example, the feature selection component only uses the InfoGain technique to select the most relevant features. Since, there exists several feature selection techniques and machine learning algorithms and their capabilities differ from each other, it would be interesting to plug in several feature selection techniques and machine learning algorithms into the feature selection and anomaly-based attack detection component. Afterward, a single feature selection technique and a single machine learning algorithm can be operationalized at a time as per the operating conditions (e.g., type of security data).

8.2.5 Impact of big data framework's configuration parameters on quality attributes

In Chapter 7, we have shown how various configuration parameters of a big data framework (i.e., Apache Spark) impact the scalability of a BDCA system. However, the impact of configuration parameters is not limited to merely the scalability of a BDCA system. An interesting avenue for future research is to investigate how configuration parameters impact other quality attributes such as CPU utilization, network utilization, disk utilization, security, modifiability, and reliability of the system. Furthermore, it would be fruitful to explore how tuning configuration parameters impact the response time of the system with respect to various contextual factors such as execution mode of the systems, the machine learning algorithm employed in the system, and the quality of the security event data. We believe that based on such exploration, a recommendation system can be developed that can recommend the optimal configuration of the framework according to the execution mode, machine learning algorithm, and security event data.

8.2.6 Industrial evaluation and feedback

Whilst the various approaches (e.g., *TACTics* and *ADABTics*) reported in this thesis have been evaluated through rigorous experimentation, it will be useful to seek feedback of practitioners and perform industry-scale evaluation of the approaches. First, such an evaluation and feedback will demonstrate the practical value of the approaches presented in this thesis. Second, the feedback can be used to improve the approaches. For instance, the guidelines for assigning weights presented in Chapter 4 are of an abstract level. These guidelines can be made more concrete based on the feedback gathered from practitioners. For industrial evaluation and feedback, we primarily suggest technical action research [270] where an industry uses the approaches in real life scenarios such as deploying our self-adaptive BDCA system for network traffic monitoring. The system can be used for a certain period of time (e.g., a month) to closely observe the performance of the system. Another potential method for seeking feedback is to carry out a case study [271] with practitioners. In the case study, the practitioners are first provided with the information about the approach (e.g., how to use *TACTics* for designing a BDCA system) and then the practitioners are asked to apply the approach. After using the approach, the practitioners can be asked various questions such as whether the practitioner found the approach useful and what are the limitations of the approach. The information collected in such a case study can help to validate and improve the various approaches presented in this thesis.

Appendix A

Selected Studies in the Systematic Literature Review

ID	System name	Title	Author(s)	Publication venue	Year
S1	Streaming-based Threat Detection	A Streaming-Based Network Monitoring and Threat Detection System	Zhijiang Chen , Hanlin Zhang , William G. Hatcher , James Nguyen , Wei Yu	Software Engineering Research, Management and Applications (SERA)	2016
S2	IP Spoofing typed DDoS detection	A Hadoop based analysis and detection model for IP Spoofing typed DDoS attack	Jian Zhang, Pin Liu, Jianbiao He, Yawei Zhang	IEEE TrustCom/BigDataSE/ISPA	2016
S3	Stochastic Self-similarity	Detecting Anomaly Teletraffic Using Stochastic Self-Similarity Based on Hadoop	JongSuk R. Lee, Sang-Kug Ye and Hae-Duck J. Jeong	Conference on Network-based Information Systems	2013
S4	Combining IDS Datasets	Combining intrusion detection datasets using MapReduce	Mondher Essid, Farah Jemili	Conference on Systems, Man, and Cybernetics	2016
S5	K-means based Anomaly Detector	Anomaly Detection in Network Traffic using K-mean clustering	R. Kumari, Sheetanshu, M. K. Singh, R. Jha, N.K. Singh	Conference on Recent Advances in Information Technology	2016
S6	Multistage Alert Correlator	Distributed Multistage Alert Correlation Architecture based on Hadoop	James Rees	International Carnahan Conference on Security Technology	2015
S7	GSLAC	GSLAC: A General Scalable and Low-overhead Alert Correlation Method	Li Cheng, Yijie Wang, Xingkong Ma and Yongjun Wang	IEEE TrustCom/BigDataSE/ISPA	2016
S8	Website IDS using Hadoop	Hadoop-based System Design for Website Intrusion Detection and Analysis	Xiaoming Zhang, Guang Wang	Conference on Smart City/SocialCom/SustainCom	2015
S9	Spark-based IDS Framework	A Framework for Fast and Efficient Cyber Security Network Intrusion Detection using Apache Spark	Govind P Guptaa, Manish Kulariyaa	Conference on Advances in Computing & Communications	2016
S10	Cloud-based Threat Detection	A Cloud Computing Based Network Monitoring and Threat Detection System for Critical Infrastructures	Zhijiang Chen, Guobin Xu, Vivek Mahalingam, Linqiang Ge, James Nguyen, Wei Yu , Chao Lu	Big Data Research	2015
S11	Spatiotemporal Correlator	A Cloud Computing Based Architecture for Cyber Security Situation Awareness	Wei Yu, Guobin Xu, Zhijiang Chen, and Paul Moulema	Workshop on Security and Privacy in cloud computing	2013
S12	Quasi Real-time IDS	Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests	Kamaldeep Singh, Sharath Chandra Guntuku, Abhishek Thakur a, Chittaranjan Hota a	Journal of Information Sciences	2014
S13	Spark-based Network Data Analysis	Network Data Analysis Using Spark	K.V. Swetha, Shiju Sathyadevan, and P. Bilna	Software Engineering in Intelligent Systems	2015
S14	Ultra-High-Speed IDS	Real time intrusion detection system for ultra-high-speed big data environments	M. Mazhar Rathore1 · Awais Ahmad1 · Anand Paul1	Journal of supercomputing	2016
S15	METRIS	METIS: A Two-Tier Intrusion Detection System for Advanced Metering Infrastructures	Vincenzo Gulisano, Magnus Almgren, and Marina Papatriantafilou	Conference on Security and Privacy in Communication Systems	2014
S16	Big Data Security Monitoring	A Big Data Architecture for Large Scale Security Monitoring	Samuel Marchal, Xiuyan Jian, Radu State, Thomas Enge	International Congress on Big Data	2014
S17	Honeypot-based Phishing Detection	A Big Data architecture for security data and its application to phishing characterization	Pedro H. B. Las-Casas, Vinicius Santos Dias, Wagner Meira Jr. and Dorgival Guedes	Conference on BigDataSecurity-HPSC-IDS	2016
S18	Malicious IP Detection	A Real-time Anomalies Detection System based on Streaming Technology	Yutan Du, Jun Liu, Fang Liu, Luying Chen	Conference on Intelligent Human-Machine Systems and Cybernetics	2014
S19	AGILIS	Distributed Attack Detection Using Agilis	Leonardo Aniello, Roberto Baldoni, Gregory Chockler, Gennady Laventman, Giorgia Lodi, and Ymir Vigfusson	Journal of Collaborative Financial Infrastructure Protection	2012

S20	Detecting DDoS in Cloud Service	A Spark-Based DDoS Attack Detection Model in Cloud Services	Jian Zhang, Yawei Zhang, Pin Liu(B), and Jianbiao He	Conference on Information security practice and experience	2016
S21	Compression model for IDS	Efficient classification using parallel and scalable compressed model and its application on intrusion detection	Tieming Chen, Xu Zhang a, Shichao Jin, Okhee Kim	Journal of Expert Systems with Applications	2014
S22	BotFinder	An Automated Bot Detection System through Honeypots for Large-Scale	Fatih Haltaş, Abdulkadir Poşul, Erkam Uzun, Bakır Emre	Conference on Cyber Conflict	2014
S23	Botnets with Graph Theory	Big Data Behavioral Analytics Meet Graph Theory: On Effective Botnet Takedowns	Elias Bou-Harb, Mourad Debbabi, and Chadi Assi	IEEE Network	2017
S24	Extreme Learning Machine	Using Extreme Learning Machine for Intrusion Detection in a Big Data Environment	Junlong Xiang, Magnus Westerlund, Dušan Sovilj, Göran Pulkkis	Workshop on Artificial Intelligent and Security	2014
S25	Intersection-based Pattern Matching	Privacy-Preserving Scanning of Big Content for Sensitive Data Exposure with MapReduce	Fang Liu, Xiaokui Shu, Danfeng (Daphne) Yao and Ali R. Butt	Conference on Data and Application Security and Privacy	2014
S26	WEKA-based Anomaly Detector	Anomaly detection model based on Hadoop platform and Weka interface	Baojiang Cui, Shanshan He	Conference on Innovative Mobile and Internet Services in Ubiquitous Computing	2016
S27	Improved K-means IDS	An Intrusion Detection System Based on Hadoop	Zhiguo Shi, Jianwei An	UIC-ATC-ScalCom-CBDCom-IoP	2015
S28	Neural-Network based DDoS Detection using Hadoop	A Neural-Network Based DDoS Detection System Using Hadoop And HBase	Teng Zhao	International Symposium on Cyberspace safety and security	2015
S29	RADISH	Detecting Insider Threats Using RADISH: A System for Real-Time Anomaly Detection in Heterogeneous Data Streams	Brock B¨ose, Bhargav Avasarala, Srikanta Tirthapura, Yung-Yu Chung, and Donald Steiner	IEEE Systems	2017
S30	Forensic Analyzer	Cloud Computing-Based Forensic Analysis for Collaborative Network Security Management System	Zhen Chen, Fuye Han, Junwei Cao, Xin Jiang, and Shuo Chen	Journal of Tsinghua Science and Technology	2013
S31	IDS-MRCPSO	MapReduce Intrusion Detection System based on a Particle Swarm Optimization Clustering Algorithm	Ibrahim Aljarah and Simone A. Ludwig	Congress on Evolutionary Computation	2013
S32	Hive vs MySQL	Performance Evaluation of Big Data Technology on Designing Big Network Traffic Data Analysis System	Nattawat Khamphakdee, Nunnapus Benjamas and Saiyan Saiyod	Conference on Soft Computing and Intelligent Systems	2016
S33	SDN DDoS Detector	A DDoS Detection and Mitigation System Framework Based on Spark and SDN	Qiao Yan(B) and Wenyao Huang	Conference on Smart Computing and Communication	2017
S34	Cloud Bursting	High-performance network traffic analysis for continuous batch intrusion detection	Ricardo Morla ·Pedro Gonçalves, Jorge G. Barbosa	Journal of supercomputing	2016
S35	GPGPU-based IDS	Design Consideration of Network Intrusion Detection System using Hadoop and GPGPU	Sanraj Rajendra Bandre, Jyoti N. Nandimath	Conference on Pervasive Computing	2015
S36	BotCloud	BotCloud: Detecting Botnets Using MapReduce	J´erˆome Franc¸ois, Shaonan Wang, Walter Bronzi, Radu State, Thomas Engel	Conference on Information Forensics and Security	2011
S37	Ctracer	Ctracer: Uncover C&C in Advanced Persistent Threats based on Scalable Framework for Enterprise Log Data	Kai-Fong Hong, Chien-Chih Chen, Yu-Ting Chiu, and Kuo-Sen Chou	Congress on Big Data	2015
S38	SEAS-MR	Scalable Security Event Aggregation for Situation Analysis	Jinoh Kim, Ilhwan Moon, Kyungil Lee, Sang C. Suh, Ikkyun Kim	Conference on Big Data Computing Service and Applications	2015
S39	APSIS	Toward a Big Data Architecture for Security Events Analytic	Laila Fetjah, Karim Benzidane, Hassan El Alloussi, Othman El Warrak, Said Jai-Andaloussi and Abderrahim Sekkaki	Conference on Cyber Security and Cloud Computing	2016
S40	Security Orchestrator	Enabling Trustworthy Spaces via Orchestrated Analytical Security	Joshua Howes, James Solderitsch, Ignatius Chen, Jonté Craighead	Workshop on Cyber Security and Information Intelligence Research	2013
S41	Hunting Attacks in the Dark	Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection	Johan Mazel, Pedro Casas, Romain Fontugne, Kensuke Fukuda and Philippe Owezarski	Journal of Network Management	2015

S42	Multi-Tier Threat Intelligence	Towards a Big Data Architecture for Facilitating Cyber Threat Intelligence	Charles Wheelus, Elias Bou-Harb, Xingquan Zhu	Conference on New Technologies, Mobility, and security	2016
S43	DDoS Detection with HTTP Packet Pattern	A method of DDoS attack detection using HTTP packet pattern and rule engine in cloud computing environment	Junho Choi ·Chang Choi, Byeongkyu Ko ·Pankoo Kim	Journal of soft computing	2014
S44	Dynamic Rule Creation based Anomaly Detector	A Dynamic Rule Creation Based Anomaly Detection Method for Identifying Security Breaches in Log Records	Jakub Breier, Jana Branis˘ova	Journal of Wireless Personal Communication	2017
S45	VALKYRIE	Valkyrie: Behavioural malware detection using global kernel-level telemetry data	Sven Krasser, Brett Meyer, Patrick Crenshaw	Workshop on Machine learning for signal processing	2015
S46	Multiple Detection Methods	Towards online anomaly detection by combining multiple detection methods and Storm	Ziyu Wang, Jiahai Yang,Hui Zhang, Chenxi Li, Shize Zhang and Hui Wang	Conference on Network Operations and Management Symposium	2016
S47	APT Detector	Study And Research of APT Detection Technology Based on Big Data Processing Architecture	Lin Shenwen, Li Yingbo, Du Xiongjie	Conference on Electronics Information and Emergency Communication	2015
S48	IDS Log Analyzer	Scalable Intrusion Detection Systems Log Analysis using Cloud Computing Infrastructure	Manish Kumar, Dr. M. Hanumanthappa	Conference on Computational Intelligence and Computing Research	2013
S49	Data Intensive Framework	Real-Time Handling of Network Monitoring Data Using a Data-Intensive Framework	Aryan Taheri Monfared, Tomasz Wiktor Wlodarczyk, Chunming Rong	Conference on Cloud Computing Technology and Science	2013
S50	PhishStorm	PhishStorm: Detecting Phishing With Streaming Analytics	Samuel Marchal, Jérôme François, Radu State, and Thomas Engel	IEEE Transactions on Network and Service Management	2014
S51	Lightweight Security Framework	Massive Distributed and Parallel Log Analysis For Organizational Security	Xiaokui Shu, John Smiy, Danfeng (Daphne) Yao, and Heshan Lin	Workshop on security and privacy in big data	2013
S53	IoT Monitor	Parallel Processing of Big Heterogeneous Data for Security Monitoring of IoT Networks	Igor Saenko, Igor Kotenko, Alexey Kushnerevich	Conference on Parallel, Distributed, and network-based processing	2017
S53	VAST	Native Actors: How to Scale Network Forensics	Matthias Vallentin, Dominik Charousset	SIGCOMM	2014
S54	Hybrid Stream and Batch Analyzer	Scalable Hybrid Stream and Hadoop Network Analysis System	Vernon K. C. Bumgardner, Victor W. Marek	Conference on Performance Engineering	2014
S55	Web IDS using Storm	Research of Recognition System of Web Intrusion Detection Based on Storm	Li Bo, Wang Jinzhen, Zhao Ping, Yan Zhongjiang, Yang Mao	Conference on Network, communication, and computing	2016
S56	Count Me In	Count Me In: Viable Distributed Summary Statistics for Securing High-Speed Networks	Johanna Amann, Seth Hall, and Robin Sommer	Journal on research in attacks, intrusions, and defences	2014
S57	PSO Clustering	Towards a Scalable Intrusion Detection System based on Parallel PSO Clustering Using MapReduce	Ibrahim Aljarah and Simone A. Ludwig	Conference on genetic and evolutionary computing	2013
S58	Dynamic Time Threshold	A Statistical Threat Detection Method Based on Dynamic Time Threshold	Jian-Wei Tian, Hong Qiao, Xi Li, Zhen Tian	Conference on Computer and Communications	2016
S59	Compound Session based Anomaly Detector	A Real-time Network Traffic Anomaly Detection System based on Storm	Gang He,Cheng Tan,Dechen Yu,Xiaochun Wu	Conference on Intelligent Human-Machine Systems and Cybernetics	2015
S60	System Call based Detector	A System Call Analysis Method with MapReduce for Malware Detection	Shun-Te Liu, Hui-ching Huang, Yi-Ming Chen	Conference on Parallel and distributed systems	2011
S61	Genetic Algo-based DDoS Detector	Distributed Denial of Services Attack Protection System with Genetic Algorithms on Hadoop Cluster Computing Framework	Masataka Mizukoshi	Congress on Evolutionary computation	2014
S62	Neural Network & Spark based DDoS Detector	Detection of DDoS attacks based on neural network using apche spark	Chang-Jung Hsieh, Ting-Yuan Chan	Conference on Applied System Innovation	2016

S63	Feature Extractor	Distributed Network Traffic Feature Extraction for a Real-time IDS	Ahmad M Karimi, Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, and Vijay K Devabhaktuni	Conference on Electro Information Technology	2016
S64	Hashdoop	Hashdoop: A MapReduce Framework for Network Anomaly Detection	Romain Fontugne, Johan Mazel, Kensuke Fukuda	Workshop on Security and Privacy in Big Data	2014
S65	ICAS	ICAS: An Inter-VM IDS Log Cloud Analysis System	Shun-Fa Yang , Wei-Yu Chen, Yao-Tsung Wang	Conf. on Cloud computing and intelligence systems	2011
S66	Batch & Stream Analyzer	Real-Time Network Anomaly Detection System Using Machine Learning	Shuai Zhao, Mayanka Chandrashekar, Yugyung Lee, Deep Medhi	Conference on the design of reliable communication networks	2015
S67	R-based Traffic Analyzer	Traffic Analysis using Hadoop Cloud	Aishwarya.K, Dr.Sharmila Sankar	Conference on Innovation in Information, Embedded, and communication systems	2015
S68	MALSPOT	MalSpot: Multi2 Malicious Network Behavior Patterns Analysis	Hing-Hao Mao1, Chung-Jung Wu1, Evangelos E. Papalexakis, Christos Faloutsos, Kuo-Chen Lee1, and Tien-Cheu	Conference on Knowledge discovery and data mining	2014
S69	P2P Bot Detector	Scalable P2P bot detection system based on network data stream	Shree Garg & Sateesh K. Peddoju & Anil K. Sarje	Journal of Peer-to-peer networking applications	2016
S70	Hadoop-based Traffic Monitor	Towards Scalable Internet Traffic Measurement and Analysis with Hadoop	Yeonhee Lee and Youngseok Lee	ACM SIGCOMM Computer Communication Review	2013
S71	Reliable Traffic Analyzer	An Internet Traffic Analysis Method with MapReduce	Youngseok Lee, Wonchul Kang, Hyeongu Son	Workshop on Network Operations and Management	2010
S72	Hybrid IDS	Design of a Hybrid Intrusion Detection System using Snort and Hadoop	Prathibha.P.G, Dileesh.E.D	Journal of Computer Applications	2013
S73	PKI-based IDS	Building Scalable Distributed Intrusion Detection Systems Based on the MapReduce Framework	Rafael Timoteo de Sousa Junior	Journal of Revista telecommunications	2011
S74	MATATABI	MATATABI: Multi-layer Threat Analysis Platform with Hadoop	Hajime Tazaki, Kazuya Okada, Yuji Sekiya, Youki Kadobayashi	Workshop on Building analysis datasets and gathering experience returns for security	2014

References

[1] F. Ullah and M. A. Babar, "Architectural Tactics for Big Data Cybersecurity Analytics Systems: A Review," *Journal of Systems and Software,* vol. 151, pp. 81-118, 2019.

[2] F. Ullah and M. A. Babar, "A Tactics-driven Approach for Designing Big Data Cyber Security Analytics Systems," *Submitted to Journal of Systems and Software (JSS),* 2020.

[3] F. Ullah and M. A. Babar, "On the Scalability of Big Data Cyber Security Analytics," *Submitted to Journal of Future Generation Computer Systems (JSS),* 2020.

[4] F. Ullah and M. A. Babar, "QuickAdapt: Scalable Adaptation for Big Data Cyber Security Analytics," in *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2019, pp. 81-86.

[5] F. Ullah and M. A. Babar, "An Architecture-driven Adaptation Approach for Big Data Cyber Security Analytics.," *International Conference on Software Architecture,* pp. 41-50, 2019.

[6] F. Ullah and M. A. Babar, "Quantifying the Impact of Design Strategies for Big Data Cyber Security Analytics Systems: An Empirical Investigation," *International Conference on Parallel and Distributed Computing, Applications, and Technologies,* 2019.

[7] J. Gontz and D. Riensel, "The Digital Universe in 2020: Big Data, Bigger Digital Shadow, and Biggest Growth in the Far East. Available at https://bit.ly/2rqPWaw [Last Accessed: 11 Feb 2020]," *IDC Country Brief,* 2012.

[8] DOMO, "Data Never Sleeps. Available at https://goo.gl/285sys [Last Accessed: 11 Feb 2020]," 2018.

[9] The-Economist, "The world's most valuable resource is no longer oil, but data. Available at https://econ.st/2Gtfztg [Last Accessed: 11 Feb 2020]," 2017.

[10] A. Oussous, F.-Z. Benjelloun, A. A. Lahcen, and S. Belfkih, "Big Data technologies: A survey," *Journal of King Saud University-Computer and Information Sciences,* vol. 30, pp. 431-448, 2018.

[11] NewVantage-Partners, "Big Data and AI Executive Survey. Available at https://bit.ly/2Y5XcZ6 [Last Accessed: 11 Feb 2020]," 2019.

[12] J. S. Ward and A. Barker, "Undefined by data: a survey of big data definitions. Available at https://bit.ly/2Dtqzam [Last Accessed: 11 Feb 2020]," 2013.

[13] C. S. Greene, J. Tan, M. Ung, J. H. Moore, and C. Cheng, "Big data bioinformatics," *Journal of cellular physiology,* vol. 229, pp. 1896-1900, 2014.

[14] P. Groves, B. Kayyali, D. Knott, and S. V. Kuiken, "The'big data'revolution in healthcare: Accelerating value and innovation," 2016.

[15] A. A. Cárdenas, P. K. Manadhata, and S. Rajan, "Big data analytics for security intelligence. Available at https://goo.gl/wxKqDV [Last Accessed: 25 November 2019]," *Big data working group,* 2013.

[16] M. Nassar, B. al Bouna, and Q. M. Malluhi, "Secure Outsourcing of Network Flow Data Analysis," *BigData Congress,* pp. 431-432, 2013.

[17] Cisco, "Cisco Visual Networking Index: Forecast and Trends. Available at https://bit.ly/2We1dq1 [Last Accessed: 11 Feb 2020]," 2019.

[18] KuppingerCole and BARC, "Big Data and Information Security: How Big Data Technology can help in increasing cyber attack resilience by better detection of attacks, enabling real-time response. Available at https://bit.ly/32O7Fqg [Last Accessed: 11 Feb 2020]," 2016.

[19] P. O. Obitade, "Big data analytics: a link between knowledge management capabilities and superior cyber protection," *Journal of Big Data,* vol. 6, p. 71, 2019.

[20] E. Chickowski, "A case study in security big data analysis," *Dark Reading,* vol. 9, 2012.

[21] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: a survey," *Journal of Big Data,* vol. 2, p. 3, 2015.

[22] Cardenas et al., "Big data analytics for security," *IEEE Security & Privacy,* vol. 11, pp. 74-76, 2013.

[23] K.-F. Hong, C.-C. Chen, Y.-T. Chiu, and K.-S. Chou, "Ctracer: uncover C&C in advanced persistent threats based on scalable framework for enterprise log data," in *2015 IEEE international congress on big data*, 2015, pp. 551-558.

[24] D. Shackleford, "Security Analytics Survey. Available at https://goo.gl/S88DTJ [Last Accessed: 11 Feb 2020]," *SANS Survey,* 2016.

[25] V. Gulisano, M. Almgren, and M. Papatriantafilou, "Metis: a two-tier intrusion detection system for advanced metering infrastructures," in *International Conference on Security and Privacy in Communication Networks*, 2014, pp. 51-68.

[26] Z. Shi and J. An, "An Intrusion Detection System Based on Hadoop," in *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, 2015, pp. 826-830.

[27] J. Rees, "Distributed multistage alert correlation architecture based on Hadoop," in *2015 International Carnahan Conference on Security Technology (ICCST)*, 2015, pp. 147-152.

[28] L. Cheng, Y. Wang, X. Ma, and Y. Wang, "GSLAC: A General Scalable and Low-Overhead Alert Correlation Method," in *2016 IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 316-323.

[29] P. H. Las-Casas, V. S. Dias, W. Meira, and D. Guedes, "A big data architecture for security data and its application to phishing characterization," in *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, 2016, pp. 36-41.

[30] S. Marchal, J. François, R. State, and T. Engel, "Phishstorm: Detecting phishing with streaming analytics," *IEEE Transactions on Network and Service Management,* vol. 11, pp. 458-471, 2014.

[31] K. Singh, S. C. Guntuku, A. Thakur, and C. Hota, "Big data analytics framework for peer-to-peer botnet detection using random forests," *Information Sciences,* vol. 278, pp. 488-497, 2014.

[32] J. Francois, S. Wang, W. Bronzi, R. State, and T. Engel, "Botcloud: Detecting botnets using mapreduce," in *2011 IEEE International Workshop on Information Forensics and Security*, 2011, pp. 1-6.

[33] L. Bass, P. C. Clements, and R. Kazman, "Software architecture in practice," *Addison-Wesley,* vol. 3rd ed., 2012.

[34] KrebsOnSecurity, "Wipro data breach. Available at https://bit.ly/2ovvCUE [Last Accessed: 11 Feb 2020]," 2019.

[35] Y. Lee and Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop," *ACM SIGCOMM Computer Communication Review,* vol. 43, pp. 5-13, 2013.

[36] L. Cheng, Y. Wang, X. Ma, and Y. Wang, "GSLAC: A General Scalable and Low-Overhead Alert Correlation Method," in *Trustcom/BigDataSE/I SPA*, 2016.

[37] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans*, et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013.

[38] Z. Chen, H. Zhang, W. G. Hatcher, J. Nguyen, and W. Yu, "A streaming-based network monitoring and threat detection system," in *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*, 2016, pp. 31-37.

[39] G. P. Gupta and M. Kulariya, "A framework for fast and efficient cyber security network intrusion detection using apache spark," *Procedia Computer Science,* vol. 93, pp. 824-831, 2016.

[40] Z. Chen, G. Xu, V. Mahalingam, L. Ge, J. Nguyen, W. Yu*, et al.*, "A cloud computing based network monitoring and threat detection system for critical infrastructures," *Big Data Research,* vol. 3, pp. 10-23, 2016.

[41] M. M. Rathore, A. Ahmad, and A. Paul, "Real time intrusion detection system for ultra-high-speed big data environments," *The Journal of Supercomputing,* vol. 72, pp. 3489-3510, 2016.

[42] Z. Chen, F. Han, J. Cao, X. Jiang, and S. Chen, "Cloud computing-based forensic analysis for collaborative network security management system," *Tsinghua science and technology,* vol. 18, pp. 40-50, 2013.

[43] S. Krasser, B. Meyer, and P. Crenshaw, "Valkyrie: Behavioral malware detection using global kernel-level telemetry data," in *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2015, pp. 1-6.

[44] J.-W. Tian, H. Qiao, X. Li, and Z. Tian, "A statistical threat detection method based on dynamic time threshold," in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, 2016, pp. 1087-1090.

[45] S. Zhao, M. Chandrashekar, Y. Lee, and D. Medhi, "Real-time network anomaly detection system using machine learning," in *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*, 2015, pp. 267-270.

[46] R. Morla, P. Gonçalves, and J. G. Barbosa, "High-performance network traffic analysis for continuous batch intrusion detection," *The Journal of Supercomputing,* vol. 72, pp. 4107-4128, 2016.

[47] I. Lytra, C. Carrillo, R. Capilla, and U. Zdun, "Quality attributes use in architecture design decision methods: research and practice," *Computing,* pp. 1-22, 2019.

[48] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord*, et al.*, "Attribute-driven design (ADD), version 2.0," Carnegie-Mellon Uni Pittsburgh PA Software Engineering Institute2006.

[49] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," in *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*, ed: sn, 2007.

[50] Apache, "Apache Hadoop. https://goo.gl/GLWG9Q [Last Accessed: 11 Feb 2020]," ed, 2009.

[51] Zaharia et al., "Apache spark: a unified engine for big data processing," *Communications of ACM,* 2016.

[52] A. Mahout, "Apache Mahout. https://goo.gl/uUVJER [Last Accessed: 11 Feb 2020]," 2017.

[53] Apache, "Spark ML Programming Guide. Available at https://bit.ly/33AIiZ3 [Last Accessed: 11 Feb 2020]," 2016.

[54] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *MSST*, 2010.

[55] KDD, "KDDcup99 Knowledge discovery in databases. https://goo.gl/Jz2Un6 [Last Accessed: 11 Feb 2020]," 1999.

[56] MIT, "DARPA intrusion detection evaluation data set. Available at https://goo.gl/jYBYNe [Last Accessed: 11 Feb 2020]," 1998.

[57] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, "Flow-based benchmark data sets for intrusion detection. Available at https://bit.ly/3ad1CQc/ [Last Accessed: 11 Feb 2020]," *ECCWS,* 2017.

[58] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. Available at https://bit.ly/30qWkft [Last Accessed: 11 Feb 2020]," *ICISSP,* 2018.

[59] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials,* vol. 18, pp. 1153-1176, 2016.

[60] S. Dua and X. Du, *Data mining and machine learning in cybersecurity*: Auerbach Publications, 2016.

[61] B. a. KuppingerCole, "Big Data and Information Security Report. Available at https://goo.gl/tffZVv [Last Acccessed: 11 Feb 2020]," 2016.

[62] G. Cybenko and C. E. Landwehr, "Security Analytics and Measurements," *IEEE Security & Privacy,* 2012.

[63] R. Verma, M. Kantarcioglu, D. Marchette, E. Leiss, and T. Solorio, "Security analytics: essential data analytics knowledge for cybersecurity professionals and students," *IEEE Security & Privacy,* vol. 13, pp. 60-65, 2015.

[64] C. E. Otero and A. Peter, "Research directions for engineering big data analytics software," *IEEE Intelligent Systems,* vol. 30, pp. 13-19, 2015.

[65] M. Shaw, "The role of design spaces.," *IEEE software 29.1,* pp. 46-50, 2012.

[66] I. Gorton and J. Klein, "Distribution, data, deployment: Software architecture convergence in big data systems," *IEEE Software,* vol. 32, pp. 78-85, 2015.

[67] G. Lewis and P. Lago, "Architectural tactics for cyber-foraging: Results of a systematic literature review," *Journal of Systems and Software,* vol. 107, pp. 158-186, 2015.

[68] G. Procaccianti, P. Lago, and S. Bevini, "A systematic literature review on energy efficiency in cloud software architectures," *Sustainable Computing: Informatics and Systems,* vol. 7, pp. 2-10, 2015.

[69] L. Garcés, A. Ampatzoglou, P. Avgeriou, and E. Y. Nakagawa, "Quality attributes and quality models for ambient assisted living software systems: A systematic mapping," *Information and Software Technology,* vol. 82, pp. 121-138, 2017.

[70] S. Mahdavi-Hezavehi, V. H. Durelli, D. Weyns, and P. Avgeriou, "A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems," *Information and Software Technology,* 2017.

[71] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Information and Software Technology,* vol. 53, pp. 625-637, 2011.

[72] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, p. 38.

[73] D. Maplesden, E. Tempero, J. Hosking, and J. C. Grundy, "Performance analysis for object-oriented software: A systematic mapping," *IEEE Transactions on Software Engineering,* vol. 41, pp. 691-710, 2015.

[74] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, 2011, pp. 275-284.

[75] A. A. C. Pratyusa and S. Rajan, "Big Data Analytics for Security Intelligence," *CLOUD SECURITY ALLIANCE,* 2013.

[76] J. Tan, S. Meng, X. Meng, and L. Zhang, "Improving reducetask data locality for sequential mapreduce jobs," in *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 1627-1635.

[77] NetFort, "Flow Analysis Versus Packet Analysis. What Should You Choose? Available at https://goo.gl/2PtAEF [Last Accessed: 11 Feb 2020]," *White Paper,* 2014.

[78] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Taxonomy and survey of collaborative intrusion detection," *ACM Computing Surveys (CSUR),* vol. 47, p. 55, 2015.

[79] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications,* vol. 36, pp. 16-24, 2013.

[80] R. Mitchell and I.-R. Chen, "A survey of intrusion detection techniques for cyber-physical systems," *ACM Computing Surveys (CSUR),* vol. 46, p. 55, 2014.

[81] K. Wiegers and J. Beatty, "Software Requirements," *Microsoft Press,* vol. Third edition, 2014.

[82] F. Losavio, L. Chirinos, N. Lévy, and A. Ramdane-Cherif, "Quality characteristics for software architecture," *Journal of Object Technology,* vol. 2, pp. 133-150, 2003.

[83] I. T. R. Center, "Breach Report. Available at https://goo.gl/oBNsPv [Last Accessed: 11 Feb 2020]," 2017.

[84] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Operational experiences with high-volume network intrusion detection," in *Proceedings of the 11th ACM conference on Computer and communications security*, 2004, pp. 2-11.

[85] Cisco, "NetFlow. Available at https://goo.gl/ko3Dyw [Last Accessed: 11 Feb 2020]," 2017.

[86] C. V. Zhou, C. Leckie, and S. Karunasekera, "A survey of coordinated attacks and collaborative intrusion detection," *Computers & Security,* vol. 29, pp. 124-140, 2010.

[87] M. Stevanovic and J. M. Pedersen, "Machine learning for identifying botnet network traffic," 2013.

[88] S. Pearson, "Taking account of privacy when designing cloud computing services," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, 2009, pp. 44-52.

[89] F. Ullah, M. Edwards, R. Ramdhany, R. Chitchyan, M. A. Babar, and A. Rashid, "Data Exfiltration: A Review of External Attack Vectors and Countermeasures," *Journal of Network and Computer Applications,* 2017.

[90] European-Commission, "Protection of Personal Data. Available at https://goo.gl/JSCxrq [Last Accessed: 11 Feb 2020]," 2016.

[91] M. Galster and P. Avgeriou, "Empirically-grounded reference architectures: a proposal," in *Proceedings of the joint ACM SIGSOFT conference--QoSA and ACM SIGSOFT symposium--ISARCS on Quality of software architectures--QoSA and architecting critical systems--ISARCS*, 2011, pp. 153-158.

[92] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," *IEEE communications surveys & tutorials,* pp. 303-336, 2014.

[93] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data," *Information Sciences,* vol. 275, pp. 314-347, 2014.

[94] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 161-168.

[95] C. Cheng, W. P. Tay, and G.-B. Huang, "Extreme learning machines for intrusion detection," in *Neural networks (IJCNN), the 2012 international joint conference on*, 2012, pp. 1-8.

[96] S. Zhao, K. Leftwich, M. Owens, F. Magrone, J. Schonemann, B. Anderson*, et al.*, "I-can-mama: Integrated campus network monitoring and management," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, 2014, pp. 1-7.

[97] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science,* vol. 315, pp. 972-976, 2007.

[98] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2016–2021. Available at https://goo.gl/WYyTV5 [Last Accessed: 11 Feb 2020]," *White Paper,* 2017.

[99] S. D. Bay, D. Kibler, M. J. Pazzani, and P. Smyth, "The UCI KDD archive of large data sets for data mining research and experimentation," *ACM SIGKDD Explorations Newsletter,* vol. 2, pp. 81-85, 2000.

[100] CAIDA, "CAIDA Dataset. Available at https://goo.gl/Tg9p8R [Last Accessed: 11 Feb 2020]," 2016.

[101] G. Combs, "TShark—Dump and Analyze Network Traffic," *Wireshark,* 2012.

[102] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM,* vol. 51, pp. 107-113, 2008.

[103] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider, "Enriching network security analysis with time travel," in *ACM SIGCOMM Computer Communication Review*, 2008, pp. 183-194.

[104] L.-B. Qiao, B.-F. Zhang, Z.-Q. Lai, and J.-S. Su, "Mining of attack models in ids alerts from network backbone by a two-stage clustering method," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, 2012, pp. 1263-1269.

[105] R. Perdisci, G. Giacinto, and F. Roli, "Alarm clustering for intrusion detection systems in computer networks," *Engineering Applications of Artificial Intelligence,* vol. 19, pp. 429-438, 2006.

[106] R. Sadoddin and A. Ghorbani, "Alert correlation survey: framework and techniques," in *Proceedings of the 2006 international conference on privacy, security and trust: bridge the gap between PST technologies and business services*, 2006, p. 37.

[107] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 245-254.

[108] M. A. Aydın, A. H. Zaim, and K. G. Ceylan, "A hybrid intrusion detection system design for computer network security," *Computers & Electrical Engineering,* vol. 35, pp. 517-526, 2009.

[109] P. Kabiri and A. A. Ghorbani, "Research on intrusion detection and response: A survey," *IJ Network Security,* vol. 1, pp. 84-102, 2005.

[110] J. F. Nieves and Y. C. Jiao, "Data clustering for anomaly detection in network intrusion detection," *Research Alliance in Math and Science,* pp. 1-12, 2009.

[111] M. Arlitt, D. Krishnamurthy, and J. Rolia, "Characterizing the scalability of a large web-based shopping system," *ACM Transactions on Internet Technology,* vol. 1, pp. 44-69, 2001.

[112] J. R. Challenger, P. Dantzig, A. Iyengar, M. S. Squillante, and L. Zhang, "Efficiently serving dynamic data at highly accessed web sites," *IEEE/ACM transactions on Networking,* vol. 12, pp. 233-246, 2004.

[113] V. Kalogeraki, P. Melliar-Smith, and L. E. Moser, "Dynamic migration algorithms for distributed object systems," in *Distributed Computing Systems, 2001. 21st International Conference on.*, 2001, pp. 119-126.

[114] S. Lewis, A. Csordas, S. Killcoyne, H. Hermjakob, M. R. Hoopmann, R. L. Moritz*, et al.*, "Hydra: a scalable proteomic search engine which utilizes the Hadoop distributed computing framework," *BMC bioinformatics,* vol. 13, p. 324, 2012.

[115] B. Pratt, J. J. Howbert, N. I. Tasman, and E. J. Nilsson, "MR-tandem: parallel X! tandem using hadoop MapReduce on amazon Web services," *Bioinformatics,* vol. 28, pp. 136-137, 2011.

[116] K. Wiley, A. Connolly, S. Krughoff, J. Gardner, M. Balazinska, B. Howe*, et al.*, "Astronomical image processing with hadoop," in *Astronomical Data Analysis Software and Systems XX*, 2011, p. 93.

[117] S. Loebman, D. Nunley, Y. Kwon, B. Howe, M. Balazinska, and J. P. Gardner, "Analyzing massive astrophysical datasets: Can Pig/Hadoop or a relational DBMS help?," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, 2009, pp. 1-10.

[118] F. Zhang, J. Cao, S. U. Khan, K. Li, and K. Hwang, "A task-level adaptive MapReduce framework for real-time streaming data in healthcare applications," *Future Generation Computer Systems,* vol. 43, pp. 149-160, 2015.

[119] T. White, "Hadoop: The Definitive Guide. O'Reilly," *Scbastopol, California,* 2009.

[120] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with MapReduce: a survey," *AcM sIGMoD Record,* vol. 40, pp. 11-20, 2012.

[121] Y. Lee, W. Kang, and Y. Lee, "A hadoop-based packet trace processing tool," in *International Workshop on Traffic Monitoring and Analysis*, 2011, pp. 51-63.

[122] I. Aljarah and S. A. Ludwig, "Parallel particle swarm optimization clustering algorithm based on mapreduce methodology," in *Nature and biologically inspired computing (NaBIC), 2012 fourth world congress on*, 2012, pp. 104-111.

[123] T. White, "Hadoop: The Definitive Guide. Availalbe at https://bit.ly/2XJe848 [Last Accessed: 11 Feb 2020]," *O'Reilly,* 2015.

[124] Cisco, "Cisco Systems NetFlow Services Export Version 9. Available at https://bit.ly/2OhSC3C [Last Accessed: 11 Feb 2020]," 2004.

[125] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing,* vol. 10, 2006.

[126] T. Bass, "Intrusion detection systems and multisensor data fusion," *Communications of the ACM,* vol. 43, pp. 99-105, 2000.

[127] I. Muttik and C. Barton, "Cloud security technologies," *Information security technical report,* vol. 14, pp. 1-6, 2009.

[128] R. Vaarandi, "Real-time classification of IDS alerts with data mining techniques," in *Military Communications Conference, 2009. MILCOM 2009. IEEE*, 2009, pp. 1-7.

[129] J. Viinikka, H. Debar, L. Mé, A. Lehikoinen, and M. Tarvainen, "Processing intrusion detection alert aggregates with time series modeling," *Information Fusion,* vol. 10, pp. 312-324, 2009.

[130] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis," *ACM transactions on information and system security (TISSEC),* vol. 6, pp. 443-471, 2003.

[131] S. Allier, N. Anquetil, A. Hora, and S. Ducasse, "A framework to compare alert ranking algorithms," in *Reverse Engineering (WCRE), 2012 19th Working Conference on*, 2012, pp. 277-285.

[132] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker, "Interactions with big data analytics," *interactions,* vol. 19, pp. 50-59, 2012.

[133] W. B. McNatt and J. M. Bieman, "Coupling of design patterns: Common practices and their benefits," in *Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International*, 2001, pp. 574-579.

[134] F. Khomh and Y.-G. Gueheneuce, "Do design patterns impact software quality positively?," in *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, 2008, pp. 274-278.

[135] F. A. Moghaddam, G. Procaccianti, G. A. Lewis, and P. Lago, "Empirical Validation of Cyber-Foraging Architectural Tactics for Surrogate Provisioning," *Journal of Systems and Software,* 2017.

[136] G. Procaccianti, H. Fernandez, and P. Lago, "Empirical evaluation of two best practices for energy-efficient software development," *Journal of Systems and Software,* vol. 117, pp. 185-198, 2016.

[137] L. Grunske, "Identifying" good" architectural design alternatives with multi-objective optimization strategies," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 849-852.

[138] S. Kim, D.-K. Kim, L. Lu, and S. Park, "Quality-driven architecture development using architectural tactics," *Journal of Systems and Software,* vol. 82, pp. 1211-1231, 2009.

[139] K. Czarnecki and U. W. Eisenecker, *Generative programming: methods, tools, and applications* vol. 16: Addison Wesley Reading, 2000.

[140] N. H. Madhavji, A. Miranskyy, and K. Kontogiannis, "Big picture of big data software engineering: with example research challenges," in *Big Data Software Engineering (BIGDSE), 2015 IEEE/ACM 1st International Workshop on*, 2015, pp. 11-14.

[141] H. T. Elshoush and I. M. Osman, "Alert correlation in collaborative intelligent intrusion detection systems—A survey," *Applied Soft Computing,* vol. 11, pp. 4349-4365, 2011.

[142] Kayacik et al., "Selecting features for intrusion detection: A feature relevance analysis on KDD 99 intrusion detection datasets," *Conference on privacy, security and trust,* 2005.

[143] T. Mahmood and U. Afzal, "Security Analytics: Big Data Analytics for cybersecurity: A review of trends, techniques and tools," in *Information assurance (ncia), 2013 2nd national conference on*, 2013, pp. 129-134.

[144] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop ecosystem," *Journal of Big Data,* vol. 2, p. 24, 2015.

[145] K. Grahn, M. Westerlund, and G. Pulkkis, "Analytics for Network Security: A Survey and Taxonomy," in *Information Fusion for Cyber-Security Analytics*, ed: Springer, 2017, pp. 175-193.

[146] L. Wang and R. Jones, "Big Data Analytics for Network Intrusion Detection: A Survey," *International Journal of Networks and Communications,* vol. 7, pp. 24-31, 2017.

[147] H.-D. J. Jeong, W. Hyun, J. Lim, and I. You, "Anomaly teletraffic intrusion detection systems on hadoop-based platforms: A survey of some problems and solutions," in *Network-Based Information Systems (NBiS), 2012 15th International Conference on*, 2012, pp. 766-770.

[148] R. Alguliyev and Y. Imamverdiyev, "Big data: big promises for information security," in *Application of Information and Communication Technologies (AICT), 2014 IEEE 8th International Conference on*, 2014, pp. 1-4.

[149] X. Zhou, Y. Jin, H. Zhang, S. Li, and X. Huang, "A map of threats to validity of systematic literature reviews in software engineering," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, 2016, pp. 153-160.

[150] T. Chen, X. Zhang, S. Jin, and O. Kim, "Efficient classification using parallel and scalable compressed model and its application on intrusion detection," *Expert Systems with Applications,* vol. 41, pp. 5972-5983, 2014.

[151] L. Bo, W. Jinzhen, Z. Ping, Y. Zhongjiang, and Y. Mao, "Research of Recognition System of Web Intrusion Detection Based on Storm," in *Proceedings of the Fifth International Conference on Network, Communication and Computing*, 2016, pp. 98-102.

[152] N. Khamphakdee, N. Benjamas, and S. Saiyod, "Performance evaluation of big data technology on designing big network traffic data analysis system," in *2016 Joint 8th International Conference on soft computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS)*, 2016, pp. 454-459.

[153] I. Aljarah and S. A. Ludwig, "Mapreduce intrusion detection system based on a particle swarm optimization clustering algorithm," in *2013 IEEE congress on evolutionary computation*, 2013, pp. 955-962.

[154] P. Prathibha and E. Dileesh, "Design of a hybrid intrusion detection system using snort and hadoop," *International journal of computer applications,* vol. 73, 2013.

[155] J. Mazel, P. Casas, R. Fontugne, K. Fukuda, and P. Owezarski, "Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection," *International Journal of Network Management,* vol. 25, pp. 283-305, 2015.

[156] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *Transactions on software engineering,* 1986.

[157] J. Mazel, P. Casas, R. Fontugne, K. Fukuda, and P. Owezarski, "Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection," *Journal of Network Management,* 2015.

[158] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Systems with Applications,* pp. 1690-1700, 2014.

[159] K. K. R. Kendall, "A database of computer attacks for the evaluation of intrusion detection systems," MIT, 1999.

[160] E. López-Iñesta, F. Grimaldo, and M. Arevalillo-Herráez, "Combining feature extraction and expansion to improve classification based similarity learning," *Pattern Recognition Letters,* vol. 93, pp. 95-103, 2017.

[161] L. Wang and R. Jones, "Big Data Analytics of Network Traffic and Attacks," in *NAECON 2018-IEEE National Aerospace and Electronics Conference*, 2018, pp. 117-123.

[162] K. Peng, V. C. Leung, and Q. Huang, "Clustering approach based on mini batch kmeans for intrusion detection system over big data," *IEEE Access,* vol. 6, pp. 11897-11906, 2018.

[163] F. Bachmann, L. Bass, and M. Klein, "Illuminating the fundamental contributors to software architecture quality," Carnegie-Mellon University, Pittsburgh, Software Engineering Institute2002.

[164] J. Ramachandran, *Designing security architecture solutions*: John Wiley & Sons, 2002.

[165] S. Kim, D.-K. Kim, L. Lu, and S.-Y. Park, "A tactic-based approach to embodying non-functional requirements into software architectures," in *2008 12th International IEEE Enterprise Distributed Object Computing Conference*, 2008, pp. 139-148.

[166] A. E. Sabry, "Decision model for software architectural tactics selection based on quality attributes requirements," *Procedia Computer Science,* vol. 65, pp. 422-431, 2015.

[167] R. Weinreich and I. Groher, "Software architecture knowledge management approaches and their support for knowledge management activities: A systematic literature review," *Information and Software Technology,* vol. 80, pp. 265-286, 2016.

[168] G. A. Lewis, P. Lago, and P. Avgeriou, "A decision model for cyber-foraging systems," *Conference on Software Architecture (WICSA),* 2016.

[169] S. Haselböck, R. Weinreich, and G. Buchgeher, "Decision guidance models for microservices: service discovery and fault tolerance," *Conference on the Engineering of Computer-Based Systems,* 2017.

[170] T. Chen, X. Zhang, S. Jin, and O. Kim, "Efficient classification using parallel and scalable compressed model and its application on intrusion detection," *Expert Systems with Applications,* 2014.

[171] Zhao et al., "I-can-mama: Integrated campus network monitoring and management," *Network Operations and Management Symposium,* 2014.

[172] J.-W. Tian, H. Qiao, X. Li, and Z. Tian, "A statistical threat detection method based on dynamic time threshold," *International Conference on Computer and Communications,* 2016.

[173] A. Paessler, "PRTG Network Monitor," ed, 2009.

[174] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," *Ieee communications surveys & tutorials,* vol. 16, pp. 303-336, 2014.

[175] N. Hubballi and V. Suryanarayanan, "False alarm minimization techniques in signature-based intrusion detection systems: A survey," *Computer Communications,* vol. 49, pp. 1-17, 2014.

[176] F. Ullah and M. A. Babar, "An Architecture-driven Adaptation Approach for Big Data Cyber Security Analytics," *International Conference on Software Architecture (ICSA),* 2019.

[177] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Systems with Applications,* vol. 41, pp. 1690-1700, 2014.

[178] J. e. a. Mazel, "Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection," *Journal of Network Management,* 2015.

[179] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Transactions on software engineering,* 2004.

[180] Russakovsky et al., "Imagenet large scale visual recognition challenge," *International journal of computer vision,* vol. 115, pp. 211-252, 2015.

[181] NCSS-Statistical-Software, "Ridge Regression. Available at https://bit.ly/3bJpLyu [Last Accessed: 11 Feb 2020]." 2016.

[182] Svahnberg et al., "A quality-driven decision-support method for identifying software architecture candidates," *Journal of Software Engineering and Knowledge Engineering,* 2003.

[183] T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi, and B. Benatallah, "A quality-driven systematic approach for architecting distributed software applications," *International conference on Software engineering,* 2005.

[184] A. MacLean, R. M. Young, V. M. Bellotti, and T. P. Moran, "Questions, options, and criteria: Elements of design space analysis," *Human–computer interaction,* vol. 6, pp. 201-250, 1991.

[185] O. Zimmermann, L. Wegmann, H. Koziolek, and T. Goldschmidt, "Architectural decision guidance across projects-problem space modeling, decision backlog management and cloud computing knowledge," *Conference on Software Architecture,* 2015.

[186] M. Soliman, M. Riebisch, and U. Zdun, "Enriching architecture knowledge with technology design decisions," *Working IEEE/IFIP Conference on Software Architecture,* 2015.

[187] M. Fahmideh and G. Beydoun, "Big data analytics architecture design—An application in manufacturing systems," *Computers & Industrial Engineering,* vol. 128, pp. 948-963, 2019.

[188] C. Wheelus, E. Bou-Harb, and X. Zhu, "Towards a big data architecture for facilitating cyber threat intelligence," *NTMS,* 2016.

[189] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: a survey," *Journal of Big Data,* 2015.

[190] G. T. Heineman and W. T. Councill, "Component-based software engineering," *Putting the pieces together, addison-westley,* 2001.

[191] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation,* 2010.

[192] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," *Future of Software Engineering,* 2007.

[193] J. Cámara, P. Correia, R. De Lemos, D. Garlan, P. Gomes, B. Schmerl*, et al.*, "Evolving an adaptive industrial software system to use architecture-based self-adaptation," *Software Engineering for Adaptive and Self-Managing Systems,* 2013.

[194] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, "Documenting software architectures: views and beyond," *ICSE,* 2003.

[195] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley, "Performance specifications and metrics for adaptive real-time systems," *Real-Time Systems Symposium,* 2000.

[196] S. Marchal, X. Jiang, and T. Engel, "A big data architecture for large scale security monitoring," *Congress on Big Data,* 2014.

[197] L. Fetjah, K. Benzidane, H. El Alloussi, O. El Warrak, S. Jai-Andaloussi, and A. Sekkaki, "Toward a big data architecture for security events analytic," *CSCloud,* 2016.

[198] R. Woolson, "Wilcoxon Signed-Rank Test," *Wiley encyclopedia of clinical trials,* 2008.

[199] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme*, et al.*, "Architecture-driven self-adaptation and self-management in robotics systems," *SEAMS'09,* 2009.

[200] M. J. Van Der Donckt, D. Weyns, M. U. Iftikhar, and R. K. Singh, "Cost-Benefit Analysis at Runtime for Self-adaptive Systems Applied to an Internet of Things Application," *ENASE,* 2018.

[201] S.-W. Cheng, D. Garlan, B. Schmerl, J. P. Sousa, B. Spitznagel, P. Steenkiste*, et al.*, "Software architecture-based adaptation for pervasive systems," *Architecture of Computing Systems,* 2002.

[202] V. Nallur and R. Bahsoon, "A decentralized self-adaptation mechanism for service-based applications in the cloud," *IEEE Transactions on Software Engineering,* 2013.

[203] M. M. Rathore, A. Ahmad, and A. Paul, "Real time intrusion detection system for ultra-high-speed big data environments," *The Journal of Supercomputing,* 2016.

[204] J. Breier and J. Branišová, "A dynamic rule creation based anomaly detection method for identifying security breaches in log records," *Wireless Personal Communications,* 2017.

[205] S. Zhao, M. Chandrashekar, Y. Lee, and D. Medhi, "Real-time network anomaly detection system using machine learning," *Design of Reliable Networks,* 2015.

[206] J. Xiang, M. Westerlund, D. Sovilj, and G. Pulkkis, "Using extreme learning machine for intrusion detection in a big data environment," *Artificial Intelligent and Security Workshop,* 2014.

[207] W. Yu, G. Xu, Z. Chen, and P. Moulema, "A cloud computing based architecture for cyber security situation awareness," *Communications Security,* 2013.

[208] P. H. Las-Casas, V. S. Dias, W. Meira, and D. Guedes, "A Big Data architecture for security data and its application to phishing characterization," 2016.

[209] K.-D. Kang and S. H. Son, "Towards security and qos optimization in real-time embedded systems," *ACM SIGBED Review,* vol. 3, pp. 29-34, 2006.

[210] T. Chen, R. Bahsoon, and G. Theodoropoulos, "Dynamic QoS optimization architecture for cloud-based DDDAS," *Procedia Computer Science,* vol. 18, pp. 1881-1890, 2013.

[211] C.-G. Lee, C.-S. Shih, and L. Sha, "Online QoS optimization using service classes in surveillance radar systems," *Real-Time Systems,* vol. 28, pp. 5-37, 2004.

[212] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi*, et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," *IEEE CVPR,* 2017.

[213] R. Asadollahi, "Starmx: A framework for developing self-managing software systems," University of Waterloo, 2009.

[214] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *Transactions on Software Engineering,* 2010.

[215] S.-N. Chuang and A. T. Chan, "Dynamic QoS adaptation for mobile middleware," *Transactions on Software Engineering,* 2008.

[216] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, "Qos-driven runtime adaptation of service oriented architectures," in *Symposium on The foundations of software engineering*, 2009.

[217] R. H. Lopes, I. Reid, and P. R. Hobson, "The two-dimensional Kolmogorov-Smirnov test," *International Workshop on Advanced Computing and Analysis Techniques in Physics Research,* 2007.

[218] L.-X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *Transactions on systems, man, and cybernetics,* vol. 22, 1992.

[219] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *Symposium on Software engineering for adaptive and self-managing systems*, 2011.

[220] T. Dahiru, "P-value, a true test of statistical significance? A cautionary note," *Annals of Ibadan postgraduate medicine,* vol. 6, pp. 21-26, 2008.

[221] F. Ullah and M. A. Babar, "Architectural Tactics for Big Data Cybersecurity Analytics Systems: A Review," *Journal of Systems and Software,* 2019.

[222] Z. C. Holcomb, *Fundamentals of descriptive statistics*, 2016.

[223] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme*, et al.*, "Architecture-driven self-adaptation and self-management in robotics systems," in *SEAMS*, 2009, pp. 142-151.

[224] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 469-482.

[225] D. N. Hill, H. Nassif, Y. Liu, A. Iyer, and S. Vishwanathan, "An efficient bandit algorithm for realtime multivariate optimization," in *SIGKDD*, 2017.

[226] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 253-266.

[227] X.-H. Sun and D. T. Rover, "Scalability of parallel algorithm-machine combinations," *IEEE Transactions on Parallel and Distributed Systems,* vol. 5, pp. 599-613, 1994.

[228] M. Kumar and M. Hanumanthappa, "Scalable intrusion detection systems log analysis using cloud computing infrastructure," in *2013 IEEE International Conference on Computational Intelligence and Computing Research*, 2013, pp. 1-4.

[229] Y. Lee and Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop," *ACM SIGCOMM Computer Communication Review,* vol. 43, pp. 5-13, 2012.

[230] J. Xiang, M. Westerlund, D. Sovilj, and G. Pulkkis, "Using extreme learning machine for intrusion detection in a big data environment," in *Proceedings of the 2014 workshop on artificial intelligent and security workshop*, 2014, pp. 73-82.

[231] I. Aljarah and S. A. Ludwig, "Towards a scalable intrusion detection system based on parallel PSO clustering using mapreduce," in *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, 2013, pp. 169-170.

[232] M. D. Holtz, B. M. David, and R. T. de Sousa Júnior, "Building scalable distributed intrusion detection systems based on the mapreduce framework," *Revista Telecommun,* vol. 13, p. 22, 2011.

[233] A. Davidson and A. Or, "Optimizing shuffle performance in spark," *University of California, Berkeley-Department of Electrical Engineering and Computer Sciences, Tech. Rep,* 2013.

[234] A. Gounaris, G. Kougka, R. Tous, C. T. Montes, and J. Torres, "Dynamic configuration of partitioning in spark applications," *IEEE Transactions on Parallel and Distributed Systems,* vol. 28, pp. 1891-1904, 2017.

[235] S. Pan, "The performance comparison of hadoop and spark," *Master Thesis. St. Cloud State University,* 2016.

[236] T. B. Perez, W. Chen, R. Ji, L. Liu, and X. Zhou, "Pets: Bottleneck-aware spark tuning with parameter ensembles," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, 2018, pp. 1-9.

[237] G. Wang, J. Xu, and B. He, "A novel method for tuning configuration parameters of spark based on machine learning," in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2016, pp. 586-593.

[238] N. Nguyen, M. M. H. Khan, and K. Wang, "Towards automatic tuning of apache spark configuration," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 417-425.

[239] A. Gounaris and J. Torres, "A methodology for spark parameter tuning," *Big data research,* vol. 11, pp. 22-32, 2018.

[240] L. Bao, X. Liu, and W. Chen, "Learning-based Automatic Parameter Tuning for Big Data Analytics Frameworks," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 181-190.

[241] K. Wang, X. Lin, and W. Tang, "Predator—An experience guided configuration optimizer for Hadoop MapReduce," in *4Th IEEE international conference on cloud computing technology and science proceedings*, 2012, pp. 419-426.

[242] S. B. Joshi, "Apache hadoop performance-tuning methodologies and best practices," in *Proceedings of the 3rd acm/spec international conference on performance engineering*, 2012, pp. 241-242.

[243] N. B. Amor, S. Benferhat, and Z. Elouedi, "Naive bayes vs decision trees in intrusion detection systems," in *Proceedings of the 2004 ACM symposium on Applied computing*, 2004, pp. 420-424.

[244] Apache, "Apache Spark. Available at https://spark.apache.org/ [Last Accessed: 11 Feb 2020]," 2014.

[245] A. Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the scalability of parallel algorithms and architectures," *IEEE Parallel & Distributed Technology: Systems & Applications,* vol. 1, pp. 12-21, 1993.

[246] X.-H. Sun, Y. Chen, and M. Wu, "Scalability of heterogeneous computing," in *2005 International Conference on Parallel Processing (ICPP'05)*, 2005, pp. 557-564.

[247] P. Jogalekar and M. Woodside, "Evaluating the scalability of distributed systems," *IEEE Transactions on parallel and distributed systems,* vol. 11, pp. 589-603, 2000.

[248] Apache, "Spark Configuration. Available at https://bit.ly/2rXR4NK. [Last Accessed: 11 Feb 2020]," 2014.

[249] Apache, "SparkHub: A Community Site for Apache Spark. Available at https://bit.ly/2lS8Vs5 [Last Accessed: 11 Feb 2020]," 2016.

[250] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu*, et al.*, "Bestconfig: tapping the performance potential of systems via automatic configuration tuning," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 338-350.

[251] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin*, et al.*, "Starfish: A Self-tuning System for Big Data Analytics," in *Cidr*, 2011, pp. 261-272.

[252] L. G. Williams and C. U. Smith, "Web Application Scalability: A Model-Based Approach," in *Int. CMG Conference*, 2004, pp. 215-226.

[253] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 363-378.

[254] Y. Du, J. Liu, F. Liu, and L. Chen, "A real-time anomalies detection system based on streaming technology," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2014 Sixth International Conference on*, 2014, pp. 275-279.

[255] P. H. Las-Casas, V. S. Dias, W. Meira, and D. Guedes, "A Big Data architecture for security data and its application to phishing characterization," in *Big Data Security on Cloud (BigDataSecurity)*, 2016, pp. 36-41.

[256] J. Kyong, J. Jeon, and S.-S. Lim, "Improving scalability of apache spark-based scale-up server through docker container-based partitioning," in *Proceedings of the 6th International Conference on Software and Computer Applications*, 2017, pp. 176-180.

[257] J. Wu, Q. Liang, and E. Bertino, "Improving scalability of software cloud for composite web services," in *2009 IEEE International Conference on Cloud Computing*, 2009, pp. 143-146.

[258] H. Senger, "Improving scalability of Bag-of-Tasks applications running on master–slave platforms," *Parallel Computing,* vol. 35, pp. 57-71, 2009.

[259] C. Canali and R. Lancellotti, "Improving scalability of cloud monitoring through PCA-based clustering of virtual machines," *Journal of Computer Science and Technology,* vol. 29, pp. 38-52, 2014.

[260] X.-H. Sun, "Scalability versus execution time in scalable systems," *Journal of Parallel and Distributed Computing,* vol. 62, pp. 173-192, 2002.

[261] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *Proceedings of the 31st International Conference on Software Engineering*, 2009, pp. 111-121.

[262] S. Tongchim and P. Chongstitvatana, "Parallel genetic algorithm with parameter adaptation," *Information Processing Letters,* vol. 82, pp. 47-54, 2002.

[263] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security,* 2019.

[264] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *Ieee Access,* vol. 5, pp. 21954-21961, 2017.

[265] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence,* vol. 2, pp. 41-50, 2018.

[266] Apache, "Deeplearning4j: Open-source distributed deep learning for the jvm," *Apache Software Foundation License,* vol. 2, 2016.

[267] M. M. Rathore, A. Ahmad, A. Paul, and S. Rho, "Urban planning and building smart cities based on the internet of things using big data analytics," *Computer Networks,* vol. 101, pp. 63-80, 2016.

[268] N. Sun, J. G. Morris, J. Xu, X. Zhu, and M. Xie, "iCARE: A framework for big data-based banking customer analytics," *IBM Journal of Research and Development,* vol. 58, pp. 4: 1-4: 9, 2014.

[269] U. Srivastava and S. Gopalkrishnan, "Impact of big data analytics on banking sector: Learning for Indian banks," *Procedia Computer Science,* vol. 50, pp. 643-652, 2015.

[270] R. J. Wieringa, *Design science methodology for information systems and software engineering*: Springer, 2014.

[271] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering,* vol. 14, p. 131, 2009.