AN ACTIVE DISTRIBUTED STORAGE ARCHITECTURE

Craig J. Patten, B.Sc. (Math. and Comp. Sc.) (Hons)

March 21, 2012

A THESIS SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE SCHOOL OF COMPUTER SCIENCE
UNIVERSITY OF ADELAIDE

Contents

Abstract **xv**

Declaration **xvii**

Acknowledgements **xix**

1 Introduction **1**
1.1 Overview 1
1.2 Motivation 1
1.3 Challenges 2
1.4 Approach 3
1.5 Contribution 5
1.6 Thesis Structure 5

2 Background and Related Work **7**
2.1 Introduction 7
2.2 Models for Data Storage and Access 8
2.2.1 Data Storage 8
2.2.2 Data Access 10
2.2.3 Summary and Relation to Current Work 11
2.3 Distributed, Extensible and Active Filesystems 12
2.3.1 Summary and Relation to Current Work 14
2.4 Distributed Systems Architectures 15
2.4.1 Structure and Communication 15
2.4.2 The Actor Model 17
2.4.3 Messaging and Enterprise Architectures 17
2.4.4 Summary and Relation to Current Work 18
2.5 Data Grids, Wide-Area I/O and Distributed High-Performance Computing 18
2.5.1 Summary and Relation to Current Work 20
2.6 Cloud Storage 20

2.7 Summary 21

3 An Active Distributed Storage Architecture 23

3.1 Introduction 23
3.2 Concepts 25
3.2.1 Core Functionality 25
3.2.2 Components 26
3.2.3 Communications 26
3.2.4 Trust and Security 27
3.3 Design and Implementation Overview 28
3.4 System Entities 30
3.4.1 Node 30
3.4.2 Components 31
3.4.3 Objects 32
3.5 Communications 32
3.6 Messages and Composition 35
3.6.1 Paths 37
3.6.2 Message Lifecycle 40
3.6.3 Data Operations 41
3.6.4 Miscellaneous I/O Operations 42
3.6.5 Component Management Operations 43
3.6.6 Custom Operations 43
3.7 Structured Data Communication 44
3.8 Component Organisation and Deployment 46
3.9 Trust and Security 50
3.10 Example: Three-Party Composition 52
3.11 Example: Implicit Paths and Path Modification 53
3.12 Component Structure 57
3.13 Summary 60

4 Components and Services 61

4.1 Introduction 61
4.2 Storage 61
4.2.1 Filesystem Interfaces: Java I/O and the Hadoop Distributed File System (HDFS) 62
4.2.2 Memory Store 65
4.2.3 Cloud Storage 65
4.2.4 Databases and Transactional I/O 67
4.3 Security 70

4.3.1 Authentication 71
4.4 Proxy 74
4.5 Transport: Parallel TCP 75
4.6 Asynchronous I/O, Write-Back Caching and Mutable Overlays 77
4.7 Atomic Writes 80
4.8 Read Caching and Prefetching 81
4.8.1 Read Caching 82
4.8.2 Prefetching 83
4.9 Distributed Key-Value Stores 84
4.10 Data Hashing, Deduplication and Content-Based Addressing 86
4.11 Message Encapsulation 88
4.12 Naming Translation 89
4.13 References 92
4.14 Service Differentiation 94
4.15 Blocking 95
4.16 Payload Transforms 98
4.16.1 Compression 98
4.16.2 Authenticated Encryption 100
4.17 Verified Storage 101
4.18 Interfaces 105
4.18.1 Web Services 106
4.18.2 File System Interface 109
4.19 Summary 112

5 Techniques and Mechanisms for Composing Distributed I/O 113
5.1 Sequential Composition 113
5.1.1 Sequential Composition with Stored Graph Elements 114
5.1.2 Dynamic Graph Generation and Injection 119
5.1.3 Third-Party Transfers 120
5.1.4 Summary 121
5.2 Distributed I/O with a Domain-Specific Embedded Language (DSEL) 121
5.2.1 Introduction 121
5.2.2 Motivation 123
5.2.3 Initial High-Level Interface 123
5.2.4 DSELs and Fluent Interfaces 126
5.2.4.1 Method Chaining 126
5.2.4.2 Nested Functions 128
5.2.4.3 Object Scoping 129

5.2.5 Fluent Interfaces and a DSEL for Distributed I/O 130
5.2.6 DSEL Compilation 137
5.3 Summary 138

6 Implementation, Case Studies and Evaluation 139
6.1 Experimental Infrastructure 139
6.1.1 Underlying Software and Hardware Infrastructure 140
6.1.2 Runtime Software Organisation 142
6.1.3 Relevance 144
6.2 Messaging Overhead 144
6.3 Basic I/O 147
6.3.1 Evaluation 148
6.4 Chained I/O 151
6.4.1 Read-then-Write 151
6.4.1.1 Evaluation 152
6.4.2 Atomic Writes or Write-then-Rename 154
6.4.2.1 Evaluation 156
6.4.3 Component Deployment 156
6.5 Multi-party I/O 157
6.5.1 Parallel TCP Streams 157
6.5.2 Write-Back Caching 163
6.5.3 Name-Based Logic: Hashing 168
6.5.4 Content-Based Logic: Deduplication 171
6.5.5 Reference Objects 175
6.5.6 Data Privacy and Integrity 176
6.5.6.1 Verified Storage 176
6.5.6.2 Authenticated Encryption 178
6.5.6.3 Evaluation 179
6.5.7 Optimistic Replication 180
6.5.8 Message Encapsulation 186
6.5.9 Read Caching and Prefetching 186
6.5.9.1 Read Caching 187
6.5.9.2 Prefetching 188
6.5.9.3 Evaluation 189
6.5.10 Service Differentiation 191
6.5.11 Aggregate Operations 192
6.5.12 Dynamically-generated I/O compositions via REST 196
6.5.13 HDFS and Topology Experimentation 198

6.6 Summary 202

7 Discussion, Future Work and Conclusion **203**

7.1 Key Points 204

7.1.1 Structure and Communication 204

7.1.2 Dynamic Composition 204

7.1.3 Flexibility and Applicability 205

7.2 Future Directions 205

7.3 Conclusion 206

A Protocol Buffer Message Specification **207**

Bibliography **211**

List of Tables

3.1 List of DARC operations and their parameters; parameter types are designated by **s** (string), **l** (long or 64-bit integer), **f** (IEEE floating-point number), **b** (byte array) and **kv** (key-value map). 39
3.2 Example component management operations. 43
4.1 Mapping between DARC operations, Java I/O and HDFS I/O functionality. 63
4.2 The custom operations offered by the BDB component, with which ACID transactions can be managed. 69
4.3 The connectors defined in the deduplication component. 86
4.4 The mapping rules for translating REST/HTTP requests to DARC messages. 109
4.5 A partial list of the mapping rules for translating filesystem requests to DARC messages. 111
5.1 The DARC DSEL vocabulary. 132

List of Figures

3.1 Local message passing. 33
3.2 Remote message passing. 34
3.3 Logical structure of a message, displaying the header and data portions. 35
3.4 Logical structure of a path, displaying the constituent metadata. 35
3.5 Hypothetical component possessing two connectors, **a** and **b**. 37
3.6 Pseudocode for the generic message processing loop at the core of component operations. 40
3.7 Example Protocol Buffer message format specification, describing a car. 45
3.8 Example use of Protocol Buffer message types defined in Figure 3.7. . 46
3.9 Example equivalent XML encoding of the `Car` object defined in Figure 3.8. 46
3.10 DARC filesystem layout; the bold elements at the lower right highlight an example component named **A** and its associated code, configuration and data. 47
3.11 An illustration of two identical dynamic component deployments via another component which possesses the code. 48
3.12 Message metadata for dynamic component deployment via another component which possesses the code. 49
3.13 Example generic security topology. 50
3.14 Original message metadata for request to restricted component. 51
3.15 Augmented message metadata for request to restricted component; new and modified fields shaded grey. 51
3.16 Topology of a simple three-way read-write-respond message. 52
3.17 Subset of path metadata for the simple three-party read-write-respond message. 53
3.18 Logical communication timeline for the simple three-party read-write-respond message. 54
3.19 Subset of original path metadata for the simple implicit composition example. 54
3.20 Subset of augmented path metadata for the simple implicit composition example; new or modified metadata is shaded grey. 55

3.21 Topology of an example simple implicit composition; original paths unmodified. 55
3.22 Topology of an example simple implicit composition; response returned from different component to that which originally received the message. 56
3.23 Subset of augmented path metadata for the simple implicit composition example; new metadata is shaded grey. 57
3.24 Example implementation of a minimal component which only responds to read requests. 58
3.25 Example modification of message metadata: directing an incoming message to an alternate location. 59
4.1 Example relationship between the HDFS interface component and other DARC components, both local and remote, which are invoking HDFS I/O operations. 62
4.2 Example path structure for remote-to-remote file copy operation. . . . 64
4.3 Timeline of message transfers in remote-to-remote file copy operation. 64
4.4 Overview of memory store. 65
4.5 Overview of cloud storage interfaces. 66
4.6 Example linkage between DARC and remote storage and compute services. 67
4.7 Example path structure for cloud-to-cloud object copy operation. . . . 68
4.8 The provision of transactional I/O via an interface to Berkeley DB; this illustration shows the initiation of a transaction by another component, using the custom `begin` operation to obtain a transaction identifier, txnID. 70
4.9 Example composition illustrating the interposition of an Authentication component between a client and a target component or subgraph. . . . 72
4.10 Example timeline illustrating the two mechanisms with which the BasicAuth component can be satisfied that a message should be allowed to continue through its path graph. 73
4.11 An illustration of the potential for transitive security relationships across network boundaries. 74
4.12 Example comparison of a single TCP connection versus the use of scattering to implement parallel TCP connection functionality. 76
4.13 Example of message path metadata modified to implement parallel TCP connection functionality; shaded path fields are new or modified. . . . 76
4.14 Example usage of two-way parallel TCP functionality. 77
4.15 Interposing asynchrony between components. 79
4.16 Providing atomic write functionality. 81

4.17 An example composition layout, illustrating a client making use of an intermediate component that maps I/O requests across a set of storage resources based on the output of a hash function. 85
4.18 Example message headers illustrating the insertion of a path redirection based on the output of a hash function. 85
4.19 Transformation of a write request by a data deduplication component. 87
4.20 Encapsulation of a message passing between components X and Y; the relation between the encapsulation and decapsulation endpoints is implicit in this example. 89
4.21 A topology showing the resolution of a name based on the joint operation of the Link component and its metadata store. 92
4.22 The path of a write request which involves a payload-encapsulated address being dereferenced in order to obtain the data which is to be written at a subsequent stage of the path graph. 93
4.23 Example usage of the Regex name-based differentiation component in a multimedia context. 95
4.24 Comparison between the external view of data presented by the Split component and the actual data organisation. 96
4.25 Example timeline for a read request traversing a blocking component interpositioned between a client and block or object store. 96
4.26 Example timeline for a write request traversing a blocking component interpositioned between a client and block or object store. 98
4.27 Comparative illustration of two different use cases for the compression component. 99
4.28 Comparative illustration of two different use cases for the encryption component. 101
4.29 Typical interposition of the Verify component, accompanied by a hash store, between a client and a downstream composition. 103
4.30 Abridged example SOAP/HTTP request for fetching an object from the Amazon S3 web service. 107
4.31 Example REST/HTTP request for fetching an object from the Amazon S3 web service. 108
4.32 The provision of external REST/HTTP access to DARC compositions. 108
4.33 The provision of conventional filesystem-style access to DARC compositions using FUSE (Filesystem in Userspace). 111
5.1 Abridged EBNF specification for pathnames used in sequential composition. 114
5.2 The entities involved in sequential pathname-based composition. . . . 115

5.3 The path entries used in the expansion of read and write operations specified by a pathname-based composition of a compression transformation and remote data store. 116
5.4 The message transfers involved in performing a read operation from a pathname-based composition of a compression transformation and a remote data store. 117
5.5 The message transfers involved in performing a write operation to a pathname-based composition of a compression transformation and a remote data store. 118
5.6 The dynamic composition and execution of chained I/O operations based on a naming scheme and request-response model. The dashed line illustrates the execution of the generated I/O graph. 119
5.7 Overview of the initial prototype for high-level access to the message-passing architecture. 124
5.8 Section of code illustrating the straightforward usage of the initial high-level DARC interface. 125
5.9 Example of the definition and usage of a simple object-oriented class definition, in this case for a car data type. 127
5.10 The concept of method chaining applied to the car example of Figure 5.9, highlighting differences in definition and usage. 127
5.11 Overview of nested functions as used within fluent interfaces; method B has two parameters, each of which is a chain of two methods, C1-C2 and D1-D2. 128
5.12 The concept of nested functions applied to the car example of Figure 5.9, highlighting differences in definition and usage. 129
5.13 The concept of object scoping applied to the car example of Figure 5.9, highlighting differences in definition and usage. 130
5.14 Software layers involved in the provision of distributed I/O functionality. 131
5.15 Using the DSEL for a third-party data transfer. 134
5.16 Using the DSEL for two parallel writes to different destinations. . . . 135
5.17 Using the DSEL for an atomic write. 136
5.18 An illustration of the levels through which compositions are dynamically generated; from Java-based DSEL down to serialised message headers. 138
6.1 Experimental infrastructure used for the tests described throughout this chapter. 143

6.2 Single-threaded message header generation performance: number of message headers generated per second versus the number of path entries in each message. 145
6.3 Single-threaded message header generation performance: bandwidth of message header generation versus the number of path entries in each message. 146
6.4 Bandwidth obtained by synchronous writes and reads directed at a DARC node located on a host within an adjacent EC2 availability zone. 149
6.5 Bandwidth obtained by asynchronous writes and reads directed at a DARC node located on a host within an adjacent EC2 availability zone. 150
6.6 I/O rate obtained by synchronous writes and reads directed at a DARC node located on a host within an adjacent EC2 availability zone. . . . 150
6.7 I/O rate obtained by asynchronous writes and reads directed at a DARC node located on a host within an adjacent EC2 availability zone. . . . 151
6.8 Bandwidth obtained when performing a read-write combination between EC2 regions us-west-2 (Oregon) and us-west-1 (California). . . 153
6.9 I/O operation rate obtained when performing a read-write combination between EC2 regions us-west-2 (Oregon) and us-west-1 (California). . 154
6.10 Three example message flows evaluated for the write-then-rename workload. 155
6.11 Comparison of component layouts for use of parallel TCP connections within a third-party data movement example. 159
6.12 Bandwidth attained in Proxy test. 162
6.13 Rate of I/O operations attained in Proxy test. 162
6.14 A moving average of the time taken per request for read and write requests with three different object sizes. 167
6.15 Rate of I/O operations attained in name-hashing test. 170
6.16 Bandwidth attained in name-hashing test. 171
6.17 A moving average of the time taken per write request when directing I/O through a local deduplication component to a distant data sink. . . 175
6.18 Bandwidth attained in the application of data verification (VERIFY) and authenticated encryption (AUTHENC) between a client and server in the us-west-1 (California) and us-west-2 (Oregon) EC2 regions respectively. 180
6.19 I/O operation rate attained in the application of data verification (VERIFY) and authenticated encryption (AUTHENC) between a client and server in the us-west-1 (California) and us-west-2 (Oregon) EC2 regions respectively. 181

6.20 Rate of I/O operations attained in optimistic replication test. 185
6.21 Bandwidth attained in optimistic replication test. 185
6.22 A random-walk read workload through a two-dimensional dataset, calculated by the Mersenne Twister pseudorandom number generator. The increasing numbers, corresponding to the colour palette, indicate the index within the workload's sequence of read operations, commencing at the origin. 190
6.23 Moving average (window size = 25 operations) of the time taken per read operation, for each of the three modes of operation, in a 500-step random walk through a two-dimensional dataset of 64 KB tile objects. 191
6.24 Bandwidth achieved from four different techniques for performing object copying from Oregon, USA to Dublin, Ireland. 195
6.25 An example data copy operation, specified by dynamic path generation, in which a client-specified compression operation is expanded to map onto one of a set of separate slave components for performing the compression. 197
6.26 Layout of DARC components for the examination of different compositions for the transfer of data from a remote data source to two mirrored HDFS clusters; each composition made use of a subset of these components. 199
6.27 A comparison of the average effective data transfer rate obtained by different compositions for sending 480 MB of data from a data source in Oregon, USA to two mirrored HDFS clusters in Virginia, USA. . . 201

Abstract

This thesis presents the Distributed Active Resource Architecture (DARC), a modular, dynamic system for the flexible construction of distributed storage and I/O infrastructure. DARC is comprised of a component-oriented framework and distributed runtime system, inspired by the Actor Model. Within DARC, components form the fundamental unit through which all functionality is provided and all communication occurs via asynchronous message-passing. This component model extends to the mechanisms through which DARC exposes its services and makes use of external storage resources. Components are modular, lightweight and can be dynamically deployed, configured and composed.

The composition of DARC components is based on directed graphs which enable arbitrary topologies of distributed I/O down to per-message granularity. These graphs are mutable through the lifecycle of each message and thus allow distributed, nested compositions. Composition is facilitated at a high level by an embedded domain-specific language for the generation and modification of graphs.

The system is implemented in Java, which provides the code mobility required for dynamic component deployment. Network communication occurs via an open platform-independent protocol, allowing interoperability. Prototype interfaces to the system include a REST-oriented web service and user-level filesystem module. Interfaces to external storage services include filesystems, object stores and cloud storage. Example functional component implementations include name- and content-based data distribution, write-back caching, parallel transfers, optimistic replication and aggregated I/O.

The implementation and evaluation described in this thesis show that the overhead of graph processing is not an impediment to high performance and illustrate the feasibility and flexibility of the design. The contribution of this work is widely applicable to a range of systems and especially dynamic cloud computing infrastructure.

Declaration

I, Craig Patten certify that this work contains no material which has been accepted for the award of any other degree or diploma 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.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

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 catalogue and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Craig J. Patten, B.Sc. (Math. and Comp. Sc.) (Hons)
June 1, 2012

Acknowledgements

First and foremost, I thank my supervisor, Fred Brown, for his guidance, support, understanding and abundant patience. His insight and advice have been central to me getting this far and are thoroughly appreciated.

I would also like to thank my original supervisor, Ken Hawick, for introducing me to the world of research, with no small amount of enthusiasm, and for opening my eyes to what is possible.

I owe a debt of gratitude to Francis Vaughan for a huge amount of support and guidance, going all the way back to day one.

I thank my colleagues at the School of Computer Science who have never ceased to be friendly, supportive and encouraging.

I thank my friends for keeping me sane and for letting me chain myself to the computer when I needed to. Special thanks to John Sharley for his thoughtful advice and for pushing me to do better.

Finally and most importantly, I thank my family, especially my parents, Joy and Don, and my wife, Aimee. Words cannot describe how much I appreciate the unbounded support which they have given me over the years. I would not have reached this point without it.