Java Coglets

Darren Webb and Andrew L. Wendelborn
Department of Computer Science
University of Adelaide
South Australia 5005, Australia

Abstract

This position paper considers how existing Grid infrastructure can be used to support Java computation on the Grid. We are interested in a class of Java applications characterized by distributed, mobile code. We identify and address these problems by building upon existing Grid services, resulting in a proposed infrastructure called Coglets that supports homogenous, secure access to Java-enabled Grid resources.

1 Introduction

The Grid is a geographically-dispersed, large-scale execution environment used to solve large, complex problems. The Grid integrates heterogeneous resources with dynamic availability and capability connected by an unreliable network. The Globus toolkit[1] is an enabling technology that provides us with a uniform view of this execution environment. The toolkit enables us to find resources, obtain performance data, stage program inputs, remotely execute programs, and stage program results.

Many projects have extended their software to take advantage of Grid-enabled resources. Generally, these projects access basic Grid services through the Java Commodity Grid toolkit (CoG)[4]. The toolkit enables their software to transfer Java code to a resource and fork a virtual machine to execute the code. An alternative is the "100% Java GRAM"[4] which treats a JAR file as an executable. The Java GRAM first stages the JAR file then forks a virtual machine to execute its code. The CoG does not address issues of locating Java-enabled Grid resources, heterogeneous Java installations, or security. The Java GRAM handles some of these issues, but necessitates replacing some existing Globus services with those of the Java GRAM.

Here we propose a new infrastructure to standardize Java computation on the Grid. This infrastructure exploits existing Globus services and protocols for securely staging and executing Java code in a manner that is natural to Java programmers. Central to this infrastructure is the *Coglet*, a simple mechanism for specifying Java execution on the Grid. Coglets aim to exploit the platform independence of Java, and the architecture-neutral protocols of Globus, in a consistent, secure manner.

2 Coglets

A Coglet is a simple specification for running Java on the Grid. A Coglet is similar to an applet, specifying a reference to Java code to be downloaded and the name of an executable class. However, Coglet execution occurs on a Grid resource. Like an applet, a Coglet is specified by a Coglet tag. This tag is an application of the Resource Specification Language (RSL), used in Globus for resource queries and job requests.

Suppose we want to issue a request to execute a Java program named `Main` whose class is stored in a JAR file named `classes.jar`, parameter `name=value` and program argument `arg1`. To execute the Java program, we start a virtual machine program with arguments including the location of classes, parameters, the name of the class containing a `main()` method, and program arguments. Specified directly in RSL it is:

```
&(executable=java)
 (arguments=-classpath classes.jar
           -Dname=value Main arg1)
```

There are several problems with this request. First, the `executable` attribute specifies the Java virtual machine program and the `arguments` attribute is littered with low-level details. The executable should be the `Main` class and its codebase in `classes.jar`. This has ramifications for executable staging. Globus supports staging of the `executable`, `stdout` and `stderr` attributes. This means the Java programmer is responsible for staging the codebase. Second, the request assumes that Java is installed, in the path, invoked by the command `java`, and is correctly configured. This is not always the case.

We propose the request be:

```
&(executable=Main)
 (codebases=https://codebase:666/classes.jar)
 (params=-Dname=value) (arguments=arg1)
```

We have introduced two new attributes and modified the interpretation of another two. The `executable` attribute now specifies a class containing a `main()` method, denoting the name of a Java program. Optionally, this attribute could specify a JAR file with a manifest entry denoting a class containing a `main()` method. The `codebases` attribute specifies the classpath as a comma-delimited list of URLs denoting the search path for classes. Each URL specifies a directory or JAR file. The `params` attribute enables the programmer to specify additional virtual machine parameters, including system properties and virtual machine configuration. Finally, the `arguments` attribute specifies input arguments for the program.

We feel the new request is more intuitive to Java developers, and better fits the definition of the request attributes. However, addition and redefinition of the request will require a specialized jobmanager capable of correctly interpreting the attributes.

We propose a *jobmanager-java* for Java programs, that interprets these request arguments and forks a machine with the correct arguments. A dedicated jobmanager for Java programs offers a number of benefits. The jobmanager-java implicitly identifies a Java bytecode processor independent of its implementation (virtual machine or direct execution processor) and low-level execution details. The jobmanager-java identifies a Java-enabled resource that can be published to the grid information services. This identifies Java-enabled Grid resources to users without the guess work. But importantly, the virtual machine program can be configured to apply security policy to code mobility. We now discuss this issue in more detail.

3 Grid Sandbox

Java introduces a number of interesting problems to the Grid. A Java class loader can download code across a network. Code downloaded across the network is not trustworthy, so there is a need to apply some control. For this purpose, Java provides the security manager or access controller to establish security policy.

One reason for the success of applets is the applet sandbox. Applets are untrusted programs downloaded over the network and run locally. The sandbox is a security manager that imposes strict controls on what untrusted code can and cannot do. For example, an applet can not access the local filesystem, execute programs or native code of the local system, or initiate socket communication to hosts other than the web server. These controls are too low-level and too restrictive for Grid computation[3].

We propose a *Grid Sandbox* that is intended to secure code mobility and still facilitates data mobility with other participants. We envisage the sandbox, built upon the Grid Security Infrastructure (GSI)[2], will establish a "ring of trust", ultimately comprising a set of mutually authenticated resources and participants within which less restrictive interchange can take place.

The sandbox enforces a policy whereby code can be downloaded only from trusted services or participants inside the sandbox. GSI provides communication integrity, hence an eavesdropper can read downloaded code but not modify it. Consequently, we do not feel it necessary to enforce strong policies that restrict file access, initiation of socket connections or invocation of external code.

4 Summary

The Coglet infrastructure we propose enables secure, peer-to-peer computation using Java on the Grid. The Coglet tag fits neatly within the RSL framework to standardize the submission of Java-based job requests. The jobmanager-java interprets the Coglet tag and initiates a Java virtual machine, with a flexible but restrictive Grid sandbox security model. The infrastructure is simple, flexible and general. Further, we believe the Coglet infrastructure can be easily applied to other commodity toolkits, such as the toolkit for Python.

References

[1] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[2] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *Proc. 5th ACM Conference on Computer and Communications Security Conference*, pages 83–92, 1998.

[3] H. Takagi, S. Matsuoka, H. Nakada, S. Sekiguchi, M. Satoh, and U. Nagashima. Ninflet: a migratable parallel objects framework using Java. *Concurrency: Practice and Experience*, 10(11–13):1063–1078, 1998.

[4] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8–9):643–662, 2001.