2002-07-01

An Enterprise-Based Grid Resource Management System

Mark J. Clement
clement@cs.byu.edu

Joseph Ekstrom

Quinn O. Snell
snell@cs.byu.edu

Kevin B. Tew

An Enterprise-Based Grid Resource Management System

Quinn Snell, Kevin Tew, Joseph Ekstrom, Mark Clement
Brigham Young University, Provo, Utah 84602
{snell, tewk, jce, clement}@cs.byu.edu

Abstract

As the Internet began its exponential growth into a global information environment, software was often unreliable, slow and had dif£culty in interoperating with other systems. Supercomputing node counts also continue to follow high growth trends. Supercomputer and grid resource management software must mature into a reliable computational platform in much the same way that web services matured for the Internet. DOGMA The Next Generation (DOGMA-NG) improves on current resource management approaches by using tested off-the-shelf enterprise technologies to build a robust, scalable, and extensible resource management platform. Distributed web service technologies constitute the core of DOGMA-NG's design and provide fault tolerance and scalability. DOGMA-NG's use of open standard web technologies and ef£cient management algorithms promises to reduce management time and accommodate the growing size of future supercomputers. The use of web technologies also provides the opportunity for a new parallel programming paradigm, enterprise web services parallel programming, that also gains bene£t from the scalable, robust component architecture.

1 Introduction

Much like the evolution of the electronic marketplace, supercomputer resource management must also evolve. Initially the web was too unreliable and insecure to adequately support electronic commerce. As standards and software evolved, companies could take advantage of a stable web server (Apache, others), secure data transmission (SSL, etc), standard database connectivity (JDBC, ODBC), and £le transfer and communication protocols (HTTP, FTP, etc.).

Supercomputer resource management can also take advantage of this evolution. Queue management can be performed through a database which yields support for transactions, persistent data, and performance. In the case of a distributed database, scalability and performance can also be attained through redundant database machines. Communication over a standard protocol such as HTTP or HTTPS yields ease in programming, secure data transfer, simple £le transfer and, where proxies are present, caching and scalability.

Supercomputing resource managers provide a way of monitoring jobs and hardware state in a supercomputing system. The resource manager starts and stops jobs and can stage data that will be used by a parallel application. When a set of supercomputers or clusters owned by different administrative authorities is used to compute the solution to a single problem, the resource managers from these different systems are used to start jobs in this metacomputer. As metacomputing systems become larger, resource managers must have the following characteristics.

1. Reliability becomes essential when multiple systems are used to solve a single problem. Bugs or failures are nearly impossible to track down across administrative domains. Most organizations have £rewalls in place between their network and the Internet. Processors from different administrative domains must be able to communicate in a reliable and secure way.

2. Scalability is important since metacomputing systems will have many more processors and jobs than single-site systems. Any system with a single point of failure or bottleneck will cause problems as the system grows. Resource managers must be able to start jobs using a "shared nothing" environment since shared memory or shared £le systems do not scale to systems that cross administrative boundaries.

3. Resource managers must be modular so that components can be replaced in order to customize behavior. Many new resource managers have been written from scratch because an existing system did not support some necessary feature.

The original Distributed Object Group Metacomputing Architecture (DOGMA) system [1, 2, 3] was developed in 1998 to address several important metacomputing issues. DOGMA provided a management system for hetero-

Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 (HPDC'02)
1082-8907/02 $17.00 © 2002 IEEE

geneous collections of workstations. It also provided for idle workstations to dynamically be used in a master-worker application just as idle nodes are utilized in the Condor [4] system. Initially, the resource manager was combined with the application and tasks were started using Java RMI. This integration of the resource manager with the application was similar to that found in the Legion system [5]. This original version of DOGMA was essentially a single user system and did not allow for native application execution.

The resource manager was separated from the application environment to create the BYU Resource Manager (YRM)[6]. This resource manager was integrated with the Maui Scheduler [7] to allow many users to run jobs at the same time in a space sharing system while still allowing for the use of volunteer idle-time workstation nodes. Several issues emerged as the system evolved that in¤uenced the design goals of DOGMA-NG.

- Security was always lacking. Custom security implementations are never as good as off the shelf web based systems such as HTTPS and SSL.
- Since the resource manager was written using custom communications transport routines, messages had to be parsed using custom code. XML, or some other standard command system are much easier to use and debug.
- Although the system was fairly reliable with 100 processors, when the system scaled up to 700 nodes, the resource manager became a bottleneck and network bandwidth was exhausted.

The DOGMA-NG resource manager addresses these problems as well as other shortcomings that have been observed in other resource managers. It is built using Commercial-Off-The-Shelf (COTS) components used in web server infrastructures. The Globus system [8] is designed to work over the Internet and recent work has focused on using web resources to discover services [9, 10]. DOGMA-NG not only uses web services, it is built out of web servers and other information infrastructure.

As redundant web servers and databases are used, the reliability and scalability can be increased to any desired level. Since the system is component based, administrators can replace security methods or select a different vendor in order to customize their implementation. The web service resource management implementation also lends itself to a different model of parallel computation where the Internet infrastructure is used to distribute and maintain information used in the parallel computation. Many applications are amenable to this type of parallel programming model. Further discussion of the web service parallel programming paradigm follows. The remainder of the paper £rst addresses the architecture of DOGMA, its bene£ts, and how it meets the needs presented above.

2 Architecture

The design of the next generation of DOGMA is based on the features needed to manage our grid system at BYU. The system must manage over 1000 idle time workstations in student labs and several small clusters between 8 and 64 nodes. It also must interact with other supercomputer schedulers that exist on our two IBM SP-2 machines. To accomplish this, the resource management system must be scalable to thousands of nodes that are connected via a wide variety of network connection speeds.

Each of the individual systems at BYU share nothing. There is no shared £le system or namespace. Scalability becomes more of an issue in a shared-nothing environment. Not only must the resource management system scale to handle all the resource information and keeping it up to date, but the distribution of program data must also be done ef£ciently. A system that pushes out the data sequentially to each node will create a bottleneck where slow interconnecting network links are concerned.

Traditional resource management systems such as PBS, and LoadLeveler are inappropriate in this environment. They each have a single point of information gathering and/or do not provide the infrastructure for a shared nothing environment. Systems like these also have specialized database and queue management engines and specialized communication protocols which have dif£culty bypassing £rewalls, etc. Our philosophy is, “Don’t reinvent the wheel–the reinvented version is never as reliable.” Let the *experts* do their job.

We have built DOGMA-NG using COTS database engines, web services, and enterprise components (J2EE) to promote standardization, pluggable and con£gurable components, and reliability. Figure 1 is a high-level portrayal of the architecture. All communication between the compute nodes and the resource management system is done via XML data over HTTP connections in a simple request/response architecture. This includes node and system information, job control, and program data. Using HTTP promotes transparent scalability and reduces bottlenecks. This is shown in Figure 1. The cluster separated from the resource management system by a slower network link in a shared nothing environment is problematic when large data £les are needed. However, a transparent proxie placed in the connecting link provides caching so that data need only traverse the slow network link once.

The DOGMA-NG system uses Enterprise Java Beans (EJB) [11] to abstract database interaction and provide for performance and scalability. Database queries are reduced to simple variable assignment and method calls that are im-

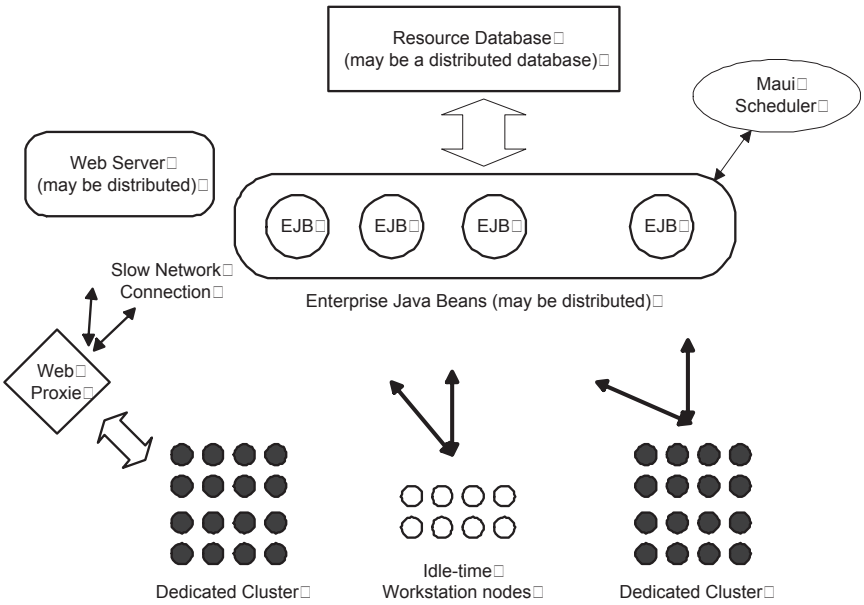

Figure 1. DOGMA-NG Architecture

plemented by the bean container in the case of container managed entity beans. J2EE containers also abstract communication, provide security and transaction management, and provide thread pooling for performance. The EJBs are pluggable components that can be easily changed, maintained, and developed. During the development phase, we have often taken down various beans and replaced others while the system and jobs remained running.

The database interaction abstraction is particularly bene£cial as database speci£c coding is not necessary. This allows for local customization of the database engine. A simple database may be used (we use PostgreSQL) or a more complex distributed database may be used for better performance. The J2EE container simply requires a Java Database Connectivity (JDBC) driver for the installed system. Such drivers are widely available for many databases. Open software and commercial databases provide stable and tested services, reducing the amount of custom code and bug potential.

The database also provides persistent storage of queue and resource information. All updates to the database are done via transactions that may be rolled back in erroneous conditions. Even when pieces of the resource management system are down or a node goes down in the process of system update, the database remains consistent. Transactions also solve synchronization issues in a highly threaded system.

Many of the interactions between the resource management system and the nodes are SOAP calls [12]. SOAP is a high-level remote procedure call mechanism that provides standardized, universal marshalling and transport. All data is marshaled in XML. Also, the endpoint services can be described in the Web Services Description Language (WSDL) [13] thus allowing language independent development and dynamic discovery of services.

Where possible, the interactions take place over an SSL connection. This includes interactions between the resource management system and the individual nodes. The encrypted SSL connection provides secure transmission of data and also secure £le transmission when needed.

Figure 2 diagrams the software architecture employed in the current version of DOGMA-NG. The architecture is based on standards, open software and a small amount of glue code. Although the system is currently running under speci£c products, the code is standards based. Thus any of the software products can be replaced with another product that implements the standard. For instance, PostgreSQL can easily be replaced with Oracle, and JBOSS can just as easily be replaced with Websphere. However, Apache, Tomcat, JBoss, and PostgreSQL provide stable, high performance services.

3 Use and Setup

3.1 Server Installation

Installation and setup of DOGMA-NG Server is relatively simple. The default installation only requires the following components:

- Java JRE 1.4

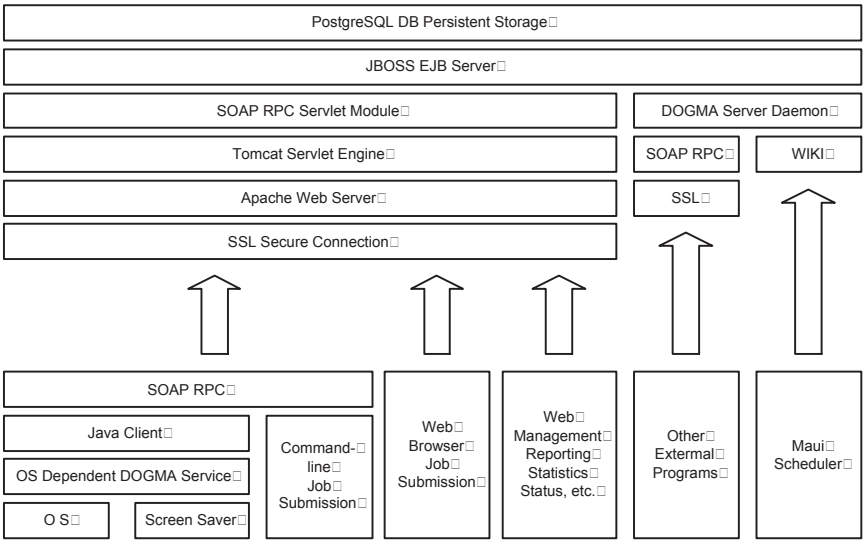

Figure 2. DOGMA-NG Software Architecture

The compile once run everywere nature of Java has made it the prefered development platform for DOGMA-NG. Although Java has its quirks and idiosyncracies, it is the most stable, platform independent, developement environment currently available. Version 1.4 of the Java class library includes useful packages such as cryptography, SSL, logging, non-blocking IO, threading, etc that allow the developer to focus his or her attention on the core issues of resource management developement instead of worring about low level communication, threading, and syncronization. Finally, Java's automated memory allocation and deallocation, frees the developer from memory management concerns and allows more rapid product development.

- Tomcat Catalina 4.0.3 [14] or another Servlet/JSP 2.3/1.2 compatible container
  The Java Servlet/JSP contianer provides SOAP object deserializtion, web based management, and http transport services. The only needed change to the Tomcat default installation is to ensure that the appropriate JNDI jars for communicating with the J2EE container are in the Tomcat classpath. This is a crucial step given that JNDI is the mechanism for obtaining EJB remote interfaces. The default DOGMA-NG installation assumes the use of the JBoss J2EE container on the localhost interface of the Tomcat server.

- JBoss 2.4.4 [15] or another J2EE version 1.3 compatible enterprise bean container
  Enterprise Java Beans provide the core services of the DOGMA-NG enterpise resource manager. BYU's DOGMA-NG uses JBoss due to its convient features such as automatic table creation and default EJB Bean to RDBMS SQL mapping. JBoss ships with the Java native Hypersonic database which can be used out of the box for small installations. The DOGMA-NG project elected to replace Hypersonic with PostgreSQL. Implementing PostgreSQL as the EJB persistant store entailed de£ning two additional JMX beans in the jboss.jcml con£g£le and including the PostgreSQL JDBC jar in JBoss' library directory.

- A RelationalDB such as PostgresSQL [16] or Oracle
  Database setup is dependent on the RDBMS used. Using PostgreSQL only requires that the inital DOGMA-NG database be created using the `createdb dogmang` command and enabling the Postgres TCP/IP listner by placing the -i argument on the postmaster command line. One last note is that the RedHat Postgres default installation only allows client database connections from the DB server localhost.

After installing the necessary industry standard servers, the DOGMA Server installation is as simple as dropping a few Web Archives (WAR) into the servlet container's webapps directory and deploying the DOGMA-NG EJB jar into the EJB container. For the more complicated environment, the JAR £les contain default properties £les that can be modi£ed to further customized and tune the DOGMA-NG installation.

3.2 Client Installation

The only prequisite for compute node installation is the Java runtime environment. Client installation simply involves downloading the platform speci£c client distribution

and running install. The compute client architecture consists of the following components:

- Native Code Daemon
  On Linux or Unix clients, the native code daemon is just a PERL script that detaches from the tty and launches java in the background. If enabled, the PERL daemon can restrict computation to during speci£c time periods during the day or whenever the machine utilization is below a certain threshold. The native code daemon on Windows clients consists of a Windows service and the DOGMA screen saver. The DOGMA screen saver, acts much like the PERL daemon, in the sense that it only permits computation when the screensaver is active. In our environment at BYU, all programs must be terminated immediately whenever the screen saver is deactivated by a user. The only exception to this rule is when the computer labs are closed.

- BootStrap Java Code
  The bootstrap Java code allows for automatic updating of the compute node software. Upon initial execution, the bootsrap code inspects the manifest of the currently installed client code and compares the local code version with the advertised version from the DOGMA-NG server. If a new version is available from the DOGMA-NG server, the client proceeds to use getURL to retreive the updated code. After retrieval and installation of the updated code, the bootstrap launches the actual compute node client.

- Compute Node Java Client
  All local resource management occurs in the compute node java client. The client uses SSL and SOAP to communicate with the global resource management server. Periodically, the compute node client updates the server with both total and currently available local system resource statistics. Finally, the client is responsible for local process creation, task launch, and task tear-down.

BYU uses DOGMA-NG to manage several clusters and a large pool of dynamic non-dedication workstations. The BYU cluster environment consists of a 48 node single processor PII 266Mhz cluster, a 16 node dual processor 733Mhz PIII cluster, and a 16 node single processor mixed PII/PIII cluster. We also use it on a new 32 node dual 933Mhz PIII / dual Athlon 2000+ cluster. Our largest resource is a pool of over a 1000 nondedicated linux and windows workstations spread across campus. Future plans call for possible integration of BYU's two IBM SP-2 supercomputers and other supercomputing systems into the DOGMA resource management system.

Jobs and Nodes in the DOGMA-NG system are currently divided into two classes dedicated and non-dedicated. Dedicated nodes include the clusters and a few nodes from the linux and windows workstation pool. These nodes are dedicated for computation. The majority of the linux and windows workstation pool fall into the non-dedicated class during the day. After the labs close at night, those same machines check-in to the system as dedicated nodes until the lab reopens in the morning. Dedicated Jobs require that all their tasks run concurrenty and without interruption, MPI jobs are one example of dedicated jobs. Non-dedicated jobs on the other hand consist of tasks that can be interrupted and do not necessarily have to run concurrently.

DOGMA-NG includes a custom round robin scheduler for scheduling non-dedicated nodes and jobs. For all dedicated nodes and jobs, DOGMA-NG uses the MAUI scheduler. This combination provides an advanced scheduling mechanism that also has the ¤exibility to use idle and unscheduled resources for dynamic jobs. Dynamic jobs may schedule resources for master programs and then request that the resource manager launch a worker process on all available idle resources. Thus, the master process is not in danger of being interrupted. Alternatively, a user may also write a web services parallel program which will be described in the next section.

To improve £le and program transfer performance in the shared-nothing environment, transparent web caching bridges have been placed at various locations. The bridge intercepts all web traf£c and sends it through the Squid proxie server. The interception does not require any changes on the cluster, lab machines, or DOGMA server. This allows the caching bridge to be transparently placed where it is needed. When a lab full of machines behind a slow connection requests a £le or program, the caching server requests the £le over the slow connection and then all other requests for it are local. We have seen dramatic decreases in startup time and network congestion using these bridges.

4 Web Service Parallel Programming

The web services architecture of DOGMA-NG presents the programmer with the opportunity to employ a new parallel programming paradigm using web services. Since the resource management system is a J2EE service, other programs can also take advantage of the J2EE architecture. Figure 3 displays a diagram of an application using this paradigm.

Given a java jar £le and a web application archive (war) containing the servlet and necessary java classes, DOGMA-NG can deploy the jar and the war, then schedule and launch worker processes. Figure 4 shows an xml job description instructing DOGMA-NG to load PhyloTreeBean.jar into the

```
<job>
 <name>Zilla</name>
 <user>snell</user>
 <type>PhylogeneticNondedicated</type>
 <class>NonDedicated</class>
 <totaltasks>10000</totaltasks>
 <maxnodes>30</maxnodes>
 <initialstate>ready</initialstate>
 <taskwallclocklimit>60000</taskwallclocklimit>
 <jobwallclocklimit>600000</jobwallclocklimit>
 <controllerjndiname>PhyloTreeBean</controllerjndiname>
 <ejbjars>
   <ejbjarurl>http://dogma-store.cs.byu.edu/Phylo/PhyloTreeBean.jar</ejbjarurl>
 </ejbjars>
 <servletwars>
   <warurl>http://dogma-store.cs.byu.edu/Phylo/PhyloTreeBean.war</warurl>
 </servletwars>

 <architectures>
   <architecture-os>
     <id>WINDOWSI86</id>
     <executable>
       <url>http://dogma-store.cs.byu.edu/Phylo/phyloclient.exe</url>
     </executable>
     <arguments>
           -cp /tmp/dogma-ng/ wrapper.PaupWrapper
           -c ./paup4b4a-x86windows -p ./zilla.nex
           -o ./qdogma -w 0.1
           -u http://clotho:8080/~snell/TestApps/servlet/PhyloTreeBeanServlet</arguments>
     <inputfiles>
       <url>http://dogma-store.cs.byu.edu/Phylo/zilla.nex</url>
     </inputfiles>
     <outputdir>
       <url>http://dogma-store.cs.byu.edu/Phylo/${jobid}/</url>
     </outputdir>
   </architecture-os>

   <architecture-os>
     <id>LINUXI86</id>

       ... similar to previous architecture-os block ...

   </architecture-os>
 </architectures>
</job>
```

Figure 4. DOGMA-NG job description for the parallel ratchet.

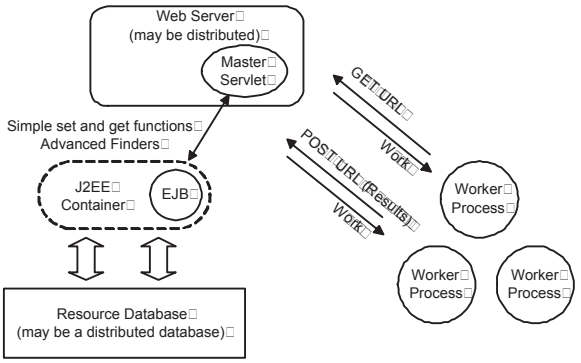

Figure 3. Web Services Parallel Programming Model

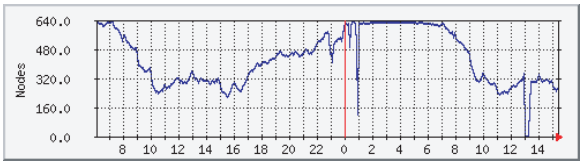

Figure 5. Number of nodes used for the parallel ratchet over a 24 hour period.

J2EE container, load PhyloTreeBean.war into the web application container, then schedule and launch 10000 total tasks for a wallclock limit of 60000 seconds each. Note that the jar and war £les need only to be accessible via URL. They can also be on the £le system local to the resource management system. When all tasks have completed the jar and war £les are undeployed. If the wallclock limit has expired, all tasks will be terminated and the jar and war £les undeployed. DOGMA-NG interfaces are currently being developed that allow applications to request more tasks and/or more time.

We have developed the parallel parsimony and likelihood ratchet for phylogenetic analysis [17] using this paradigm. Because of its detached communication, not only is it suited for general parallel machines, but it also is ideal for idle time processing. In our environment at BYU, jobs on the idle-time workstations must be immediately terminated when a student begins using a machine. The web services allow machines to come and go without any specialized coding in the master process (HTTP and the web server already deal with this). Figure 5 shows a graph of the number nodes used over time for a run of the parallel ratchet.

Another advantage of this system is that results are automatically persisted to the database. Thus the application can be restarted from where it left off in the event of machine failures or time limit expiration. For extremely long jobs, this promotes fair use of the computing resources. Web service based jobs can be scheduled for shorter time periods and then rescheduled to allow other jobs time on the system.

5 Conclusion

As systems continue to grow and workstations become available for dynamic grid computing, reliability fault tolerance, and scalability in a resource management system becomes more important. In this paper, we have described the DOGMA-NG resource management system. DOGMA-NG is a highly scalable resource management system built on tested COTS components. Together, these components and the DOGMA-NG system form a robust and extensible grid resource management system. The web services and enterprise components which constitute the core of the system provide fault tolerance as well as high performance and scalability.

The DOGMA-NG system also provides the opportunity for web services parallel programming. This is a very reliable and fault tolerant paradigm for master-worker parallel applications. Using this model, we have been very succesful at achieving high performance on large numbers of nodes.

References

[1] Glenn Judd, Mark Clement, and Quinn Snell. DOGMA: Distributed object group metacomputing architecture. *Concurrency Practice and Experience*, 10(1):1–7, 1998.

[2] Quinn Snell, Glenn Judd, and Mark Clement. The DOGMA approach to parallel and distributed computing. *Parallel and Distributed Computing Practices*, 2(2), June 1999.

[3] Glenn Judd, Mark J. Clement, and Quinn O. Snell. The DOGMA approach to high-utilization supercomputing. In *Proceedings of the Seventh Internatational Symposium on High Performance Distributed Computing (HPDC-7), Chicago, Illinois*, 1998.

[4] Michael J. Litzkow, Miron Livny, and Matt W. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the Eighth Internatational Conference on Distributed Computer Systems*, 1988.

[5] A Grimshaw et. al. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), 1997.

[6] Daniel L. Reese, Scott V. Hansen, Quinn O. Snell, and Mark J. Clement. YRM: An advanced resource man-

ager. In *Proceedings of the Conference on Parallel and Distributed Computing Systems (PDCS)*, 2000.

[7] David Jackson, Quinn Snell, and Mark Clement. Core algorithms of the maui scheduler. *Job Scheduling Strategies for Parallel Processing, (LNCS) Editors: Dror G. Feitelson and Larry Rudolph*, 2221, June 2001.

[8] Ian Foster and Carl Kesselman. *Globus: A Toolkit-Based Grid Architecture*. Morgan Kaufmann, 1999.

[9] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal on Supercomputer Applications*, 15(3), 2001.

[10] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid. http://www.globus.org/research/papers/ogsa.pdf.

[11] Java 2 platform enterprise edition speci£cation, v1.3. http://java.sun.com/j2ee.

[12] Simple object access protocol (SOAP) W3C. http://www.w3.org/2002/ws/.

[13] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language 1.1, W3C note 15 march 2001. http://www.w3.org/TR/wsdl.

[14] Jakarta tomcat. http://jakarta.apache.org.

[15] Jboss. http://www.jboss.org.

[16] Postgresql. http://www.postgresql.org.

[17] Quinn Snell, Michael Whiting, Mark Clement, and David McLaughlin. Parallel phylogenetic inference. In *Proceedings of Supercomputing 2000, Dallas, Texas*, November, 2000.