Taking Dependencies into Account in Grid Resource Allocation

Sean. Norman, Michael Katchabaw, Hanan Lutfiyya
Department of ComputerScience
The University of Western Ontario
London,Ontario Canada
{snorman,katchab,hanan}@csd.uwo.ca

Abstract

This paper describes the current work on developing a generic framework and a prototype for resource selection in a grid computing environment that addresses the problems found in other work. Specifically, this work allows for customized application-specific mapping strategies that provide information on the dependencies that an application may have on certain resources or applications.

Keywords

[grid computing, resource management]

1. Introduction

Grid computing refers to the use and coordination of dispersed computing resources that includes servers, workstations, desktops, storage, etc. The concepts inherent to grid computing were initially used in numerically intensive scientific computing applications e.g., [1]. Recently there has been the desire to develop grids for commerce, based on the premise that a grid enables more efficient sharing and utilisation of computing power and storage. It has been estimated that commercial computing resources reach only 10% or 20% utilisation of their full potential [12], while sitting idle much of the time. Such idle resources could be used for applications that demand large computing power, as illustrated in the desktop realm with the SETI@Home and Folding@Home projects [1]. IBM and Charles Schwab's Advanced Technology Group recently announced that it was able to reduce the processing time of an existing financial application from more than four minutes to 15 seconds by grid enabling it with the Globus Toolkit running RedHat Linux on IBM xSeries 330 machines [15].

The allocation and scheduling of computing resources is known as *resource management*, hereafter referred to as management. A *job* refers to the desired

execution of an application a_i that consists of a group of distributed components, (which are essentially executable files) a_{i0},a_{i1}, …,a_{ir} , that work together to accomplish a single goal. A *resource configuration* is defined as the allocation of computing resources to an instance of a_i, which is essentially one or more instantations of a_{i0},a_{i1}, …,a_{ir} . An instantiation is referred to as a *job.*

Determining a resource configuration depends on criteria that includes the following: (i) Requirements of the resources needed for an application; For example, an application component may require a minimum amount of disk space, memory, CPU speed, etc; (ii) The placement of application components with respect to each other. For example, an application component may need to read heavily from a database server. This suggests that the number of hops between the application component and the database server is minimized; and (iii) System level requirements that refer to requirements that are not necessarily associated with a specific application. For example, two applications that consume a large amount of bandwidth would not likely be deployed so they compete for the same network links.

Efficient management of resources also requires that it is possible to keep track of the resource configuration for an instance of an application that is available to user of the application and the management infrastructures. Some systems do allow applications to keep track through scripts but this is not systematic. Having the infrastructure maintain information on the resource configurations allows for a uniform interface and provides support for balancing load.

To summarize, there are two requirements that is needed for a grid management system: (i) The ability to use information about dependencies to place executables; and (ii) The ability to keep track of the locations that an application component is instantiated on.

This paper describes an application deployment framework and its relationship to existing grid standards that addresses the requirements discussed in the previous paragraph for enterprise grids. The paper is organized as follows: Section 2 describes related work. Section 3 discusses the architectural framework of the proposed resource management infrastructure. Section 4 discusses the implementation. Section 5 provides a discussion of the framework. Conclusions are presented in Section 6.

2. Related Work

Many projects have attempted to address the problem of resource selection. LSF [16], LoadLever [10], NQE [5] and PBS [9] are older systems that process user jobs by finding resources that have been identified either explicitly through a job control language or implicitly by submitting a job to the queue associated with a set of

particular resources [2]. One of the disadvantages of this approach is that it requires users to be knowledgeable about specific system resources and requires manual configuration of the queues, which does not lend itself to dynamic resource discovery. The Globus [6], Legion [4] and Condor [11] projects provide infrastructures that support dynamic resource discovery, resource allocation, job control and dynamic resource monitoring. Legion provides a generic, default scheduler which can be extended or replaced in order to take into account application-specific information. The AppLeS framework [3] utilizes an application-centric approach whereby it is more important to promote the performance of an individual application rather t han optimize the use of system resources or maximize the throughput of a collection of jobs [3]. Utilization of AppLeS is tedious because it requires each application to be manually configured via the use of a graphical user interface. A general-purpose resource selection framework, proposed in [2], defines a resource selection service for locating grid resources that match application requirements. The framework extends Condor matchmaking mechanisms to support both single and multiple resource selections, and provides an open interface for loading application-specific mapping modules to personalize the resource selector [2].

While these projects address some of the issues related to resource selection, only AppLeS and the resource selection framework proposed in [2] allow for customized, application-specific mapping strategies. Customized mapping strategies are necessary to applications have dependencies on certain resources or applications.

However, one of the problems with the mapping strategy used in [2] is that it requires the end-user to write a mapping module specific to that application. In order to specify these types of mappings, the underlying framework must support both single and multiple resource selection, much like the Condor extensions proposed in [2] or Globus DUROC mechanisms [8].

Of the few projects that utilize application deployment mappings, none keep track of the mappings for the purposes of managing current and future application deployments. For example, a distributed application that requires high bandwidth links between its nodes and has low latency requirements might need to be deployed on nodes that are not running similar types of applications. Thus, the resources on which applications are deployed become an important scheduling decision when selecting resources for new application deployments. A framework that takes this information into account may provide further management services, such as a service that monitors application deployment and re-deploys the applications or parts of it when performance degradation exceeds a specific threshold. A framework such as the one proposed in [2] has the fundamental concepts that could support this type of idea, but the fact that the framework is tightly integrated with Condor makes this impossible; Condor assumes that a single job or sub-job utilizes an entire resource. Thus, only one job can run on a single resource at any given time. Although it is possible to specify that a job requires a certain number of resources, each of these jobs or sub-jobs will fully utilize each of the resources, removing them from the pool

of available resources (although pre-emption is possible).

3. Framework

The development of the framework was based on requirements that included the following:

- It should be possible to support *intelligent* deployment decisions taking into consideration resource availability, application dependencies, and system level requirements.
- Information about an application's resource configuration should be maintained and an interface provided so that it is possible to retrieve this information by management applications and the submitter of the job subject to security constraints.
- Significant overhead on the grid or the applications executing within the grid should not be incurred.
- The management infrastructure must be scalable to support large grid environments.
- Constraints or restrictions should not be imposed on the grid or its applications. At least from an architectural level, it should be portable and platform-neutral.
- The framework must be flexible, configuration and easily extendible to make it easy to deploy itself and be accommodated in existing systems, and to permit future growth and development.

This section describes the framework of an *application deployment system* [13] that receives a job request, finds the appropriate resources and deploys the application to the appropriate resources. The framework is based on the assumption that web services are used and is graphically depicted in Figure 1.

3.1 Job Client

This component is the interface to the application deployment system. It may on any number of machines. The execution of an application requires the following information: (i) Executables, and input and script files needed for the actual execution of the application; (ii) Requirements of execution environment which is specified as constraints on the type and amount of a specific resource. For example, an application component, a_{ij}, may require that it executes under the Windows operating system, that there is at least 1 GB of memory available, etc; Generally resources that contraints may be placed on includes the architecture, operating system, mimimum memory needed, and amount of disk space needed. This is close to what is allowed to be specified using resource specification languages such as the Job Submission Definition Language (JSDL).

For any two components of application a_i, a_{ij} and a_{ik}, it should be possible to specify the *weight* of the link. In this context, the weight is a measurement of the amount of communication that is expected to take place between a_{ij} and a_{ik}. A lower weight

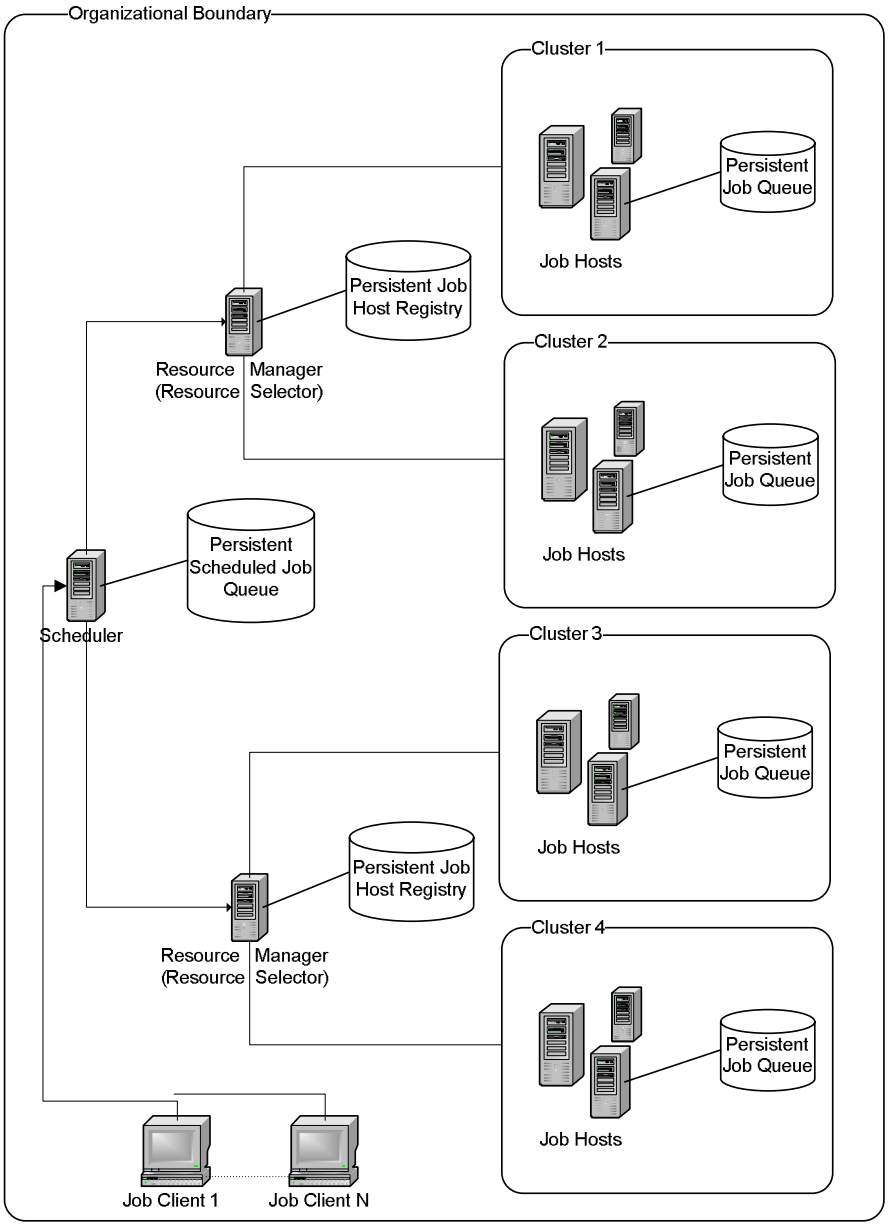

Figure 1: Application Deployment Framework

implies that the network latency between a_{ij} and a_{ik} should be as minimal as possible. This is a new feature not currently found in existing resource specification languages. To illustrate the usefulness of this information consider the following example parallel applications. Parallel programs supporting Monte Carlo simulations are organized such that the application components do not have to communicate with each other. Thus, the placement of the application components does not need to take into account network resources. On the other hand, parallel applications that solve a large system of linear equations are tightly-coupled as well as compute and network intensive.

This suggests that application components should be placed relatively close to each other. It is not always the case that the desired resources are found. We allow for the possibility of specifying alternative requirements for an execution environment. If it is not possible to satisfy any of the alternative requirements then an error is returned to the *Job Client* component indicating this.

An example job description file is found in Figure 2. In a job description file, information about an application component's deployment requirements is specified in between a set of square brackets. The first line has the tag GROUP_ID that uniquely identifies this application component from other application components in the job description file. In the example found in Figure 2, the information in between the first set of square brackets specifies two instances of a master component are needed that execute on a Microsoft Windows-based machine. The line with the tag TYPE refers to the type being executable being submitted. Job types include *windows executable, windows/command, java/jar, java/class, linux/shell, linux/binary, solaris/shell, solaris/binary.* The tag EXEC specifies a path to the executable file. An optional second parameter specifies the relative path that the executable should be deployed on the target job host. The tag CONSTRAINT specifies constraints on the job hosts that this job will be deployed on. Attributes that can be used include job host name, vendor, cpu load, free memory, idle time, etc; Multiple constraints are specified in order to show that components can be deployed as long as one set of constraints can be satisfied. For example, the first two sets of constraints described for the application component, "MASTER 1", will fail if the environment does not contain Microsoft Windows 95 or 98 machines. The tag INPUT_ARG specifies input arguments to pass to the executable when it starts. The tag INPUT_FILE represents a path to an input file required by the executable. If there are multiple input files then there are multiple lines with the INPUT_FILE tag. The tags, MIN_DEPLOYMENTS and MAX_DEPLOYMENTS are used to specify the minimum number and minimum number of job hosts needed.

```
[
GROUP_ID   "MASTER1"
TYPE               "windows/executable"
EXEC               "/usr/gridapp/gridapp.exe"
INPUT_ARG          "true" "300"
CONSTRAINTS        "(&(jhosname=MicrosoftWindows)(jhosarch=i386))"
CONSTRAINTS        "(&(jhosname=MicrosoftWindows
98)(jhosarch=i386))"
CONSTRAINTS        "(&(jhosname=MicrosoftWindows
XP)(jhosarch=i386))"
MAX_DEPLOYMENTS   "2"
MIN_DEPLOYMENTS   "2"
DIST                       "GROUP1" "1" "GROUP1" "2" "1"
DIST                       "GROUP1" "1" "GROUP2" "1" "1"
DIST                       "GROUP1" "1" "GROUP2" "2" "1"
DIST                       "GROUP1" "2" "GROUP2" "1" "1"
DIST                       "GROUP1" "2" "GROUP2" "2" "1"
DIST                       "GROUP2" "1" "GROUP2" "2" "1"
]
[
GROUP_ID          "MASTER2"
TYPE                       "linux/binary"
EXEC                       "/usr/gridapp/gridapp.bin"
INPUT_ARG                  "true" "300"
CONSTRAINTS                "(&(jhosname=Linux)(jhosarch=i386))"
MAX_DEPLOYMENTS   "2"
MIN_DEPLOYMENTS   "2"
]
[
GROUP_ID          "SLAVES"
TYPE                       "java/jar"
EXEC                       "/usr/gridapp/gridapp.jar"
INPUT_ARG                  "true" "300"
INPUT_FILE                 “/usr/gridapp/data/test.dat” “/data”
CONSTRAINTS                "(&(jhosname=*)(jhosarch=i386))"
MAX_DEPLOYMENTS   "100"
```

Figure 2: **Sample Job Description File**

The tag DIST specifies the maximum distance allowed between jobs running the specified jobs. Each line is specified as follows: *source group identifier*, *source job number*, *destination group identifier*, and *destination job number*. In the current implementation, distance is defined as the number of hops. Future work will examine other measures of distance.

3.2 Resource Manager

The Resource Manager component provides information about the available computing resources. It maintains both static and dynamic information about computing resources. Static information about a workstation includes the operating system it is using, CPU speed, the subnetwork it is on, etc; Dynamic information about a workstation includes the CPU utilization. Also maintained is information

regarding the relationship between different subnetworks. The Resource Manager allows proxies on behalf of computing resources to register static information about it that is stored in a repository. Dynamic information may be collected using a variety of monitoring tools. For example, CPU availability may be measured using an SNMP agent while a network weather service may be used to monitor bandwidth utilization.

3.3 Scheduler

The Selector receives an application deployment request and deploys the application files (e.g., executables, scripts) to the chosen computing resources. To do this, the scheduler parses the application deployment request and makes a list of the computing resources needed. It then submits a request to a resource manager that specifies a list of required resources and the requirements of these computing resources. The Resource Manager returns a list of available resources. This is provided to the Resource Selector component. The choice of resources is not necessarily based only on the application deployment request. It may also be based on system policies. Once the selection of resources is completed and provided to the Selector, the Scheduler then deploys the application components. The Scheduler assigns job identifier numbers. The Scheduler maintains information about the mapping of the application to the chosen computing resources. There is an interface that can be used by the client and the Resource Selector for querying application deployment and configuration information.

3.4 Resource Selector

This component is provided with a possible set of resources. It is a subcomponent of the Resource Manager. This component selects from these resources. It is not possible to evaluate every possible combination since this is expontential with respect to the number of resources. We combine the use of a heuristic algorithm with the use of policies to determine a reasonable set of resources. If there are no constraints associated with communication then for each application component the first returned resource is chosen for an application component is returned. Policies can be applied to filter out resources. Examples include the following: (i) There may be access control policies that do not allow the user to have access to a specific resource; (ii) Another policy may have a limit on of applications allowed to execute on a specific machine or use the same local area network. This eliminates resources from being considered. Policies take the form *if* **cond** *then* **actions.** There is existing software that can support the implementation of policies e.g., JESS[12].

3.5 Job Host

The job host software consists of the following: (i) A proxy for registering the host machine with a resource manager and provides resource usage information; and (ii) A process that keeps track of the job. The start time and end time of a job is logged to a database. The process may delay the start of a job. An interface is provided that allows other management infrastructure components to get at this information.

3.6 Interactions

This section briefly describes the interactions between the different components. An example of a possible interaction is graphically depicted in Figure 3. A user submits an application to the management infrastructure. This consists of a job description file that refers to locations of the executables and input files. The request is passed to the Scheduler. The Scheduler sends a request to the Resource Manager. For each application component found in a group, the Resource Manager finds all the machines that satisfy the constraints. The Resource Selector (a component of the Resource Manager) makes the final selection based on the policies. The results are returned to the Scheduler. The Scheduler then sends the executable and its input files to the machines that were assigned to the executable. The results are returned to the machine that the job was submitted from.

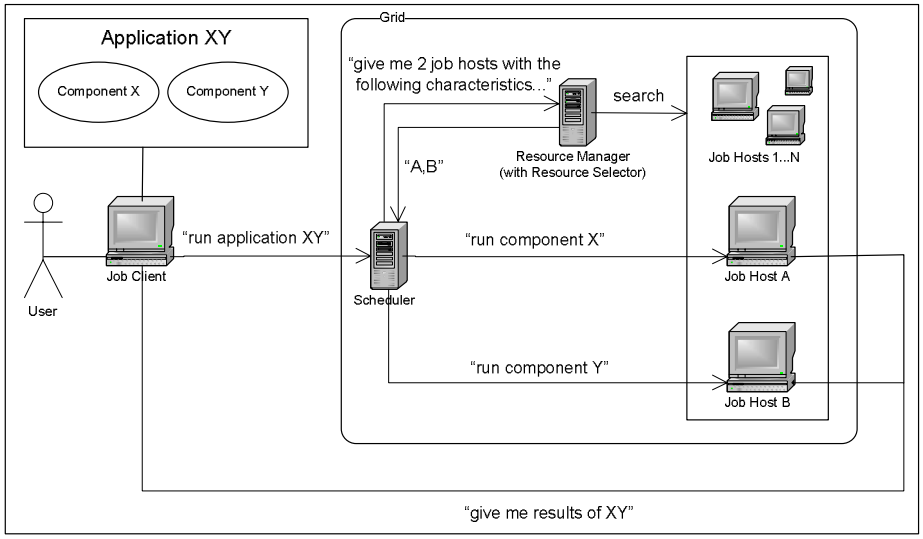

Figure 3: Interactions in Application Deployment

4. Implementation

An initial prototype has been developed. Each of the components of the infrastructure (except the job client) consists of a servlet container, a repository and a local service. The servlet container hosts the web service. This organization allows for a good deal of flexibility in deploying the infrastructure. For example, the Job Host servlet container, database server and local service can each run on the same machine or be distributed amongst different machines. The same servlet container and repository can be shared. All repositories used DB/2 while the Resource

Manager's repository used LDAP. The web service provides an interface to the component logic.

5. Discussion

In this section we discuss the framework's satisfaction of requirements identified earlier as well as our current observations.

Support for Intelligent Deployment Decisions. The ability to support the selection of resources based on resource usage, policy and application needs is supported through the Resource Selector. The Resource Selector uses a rule-based engine to implement policies. This allows for an approach to selecting resources based on system requirements.

Support for Application's Resource Configuration Maintenance. This is supported by the Scheduler. The web service provides an interface that allows for the retrieval of information about resource configurations.

Must be Scalable. More work needs to be done to validate that this is scalable. Scalability will be achieved by having having multiple schedulers for different domains. A user will submit a job to the "local" scheduler. If the "local" scheduler cannot find all of the necessary resources within its domain, it then sends a request for resources to neighboring schedulers. These in turn may forward the request to other schedulers. Two issues need to be addressed: (i) How much forwarding should be allowed? and (ii) It may be the case that there some local resources. The scheduler needs to partition the application request appropriately so that it takes into account application needs. For example, if for two components the number of hops should be one and there are local resources to satisfy this, then the local scheduler will request resources for the rest of the application .

Must be Flexible and Easy to Extend. This work relies on the use of web service technology. The web service provides for a standard interface. Thus, different domains may have different implementations of the different components of the management architecture proposed in this paper. For example, different domains may implement the scheduler based on the use of an existing scheduler.

The Resource Selector uses a rule-based engine to implement policies. A change in policies can be mapped to new rules to be loaded by the rule-based engine. This allows for a change in policies without recompiling the code. Additionally, this also allows for different appraoches and languages to be provided to administrators for specifying policies. The only requirement is that there is a mapping from the policy that the administrators specify to the rules that can be read by a rule engine.

Adding resources is easy. When a new computing resource is available, the software for registration can be put on that device. That device can then register with the Resource Manager.

There are several limitations with respect to flexibility including the following: (i) Currently, the job description language we have specifies the minimum and maximum

number of deployments of an application. Ideally, the management system would determine this number based on the performance requirements of the application; (ii) Currently, results are returned to the host machine where the submission was made. It should be possible to specify in the job description file a different target to return results.

6. Conclusions

This paper demonstrates that it is possible to specify application dependencies and use this information in selecting resources. The framework was presented. Discussion included advantages of the presented framework and disadvantages. Future work will address these disadvantages and look at placing the software in a larger environment.

ACKNOWLEDGMENT

We would like to thank Natural Sciences and Engineering Research Council {NSERC) of Canada. for their support.

References

1. Anderson, D., Cobb, J., Korpela, M., Lebofsky, M., and Werthimer, D., *SETI@home: An Experiment in Public-Resource Computing*. Communications of the ACM, Volume 45, No 11.
2. Angulo, D., Foster, I., Liu, C. and Yang, L. *Design and Evaluation of a Resource Selection Framework for Grid Applications*. Proc. of IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July, 2002. <http://www.globus.org/research/papers/RS-hpdc.pdf>
3. Berman, F. and Wolski, R. *The AppLeS project: A status report*. Proc. of the 8th NEC research Symposium, 1997. <http://www.cs.ucsd.edu/groups/hpcl/apples/pubs/nec97.ps>
4. Chapin, S., Grimshaw, A., Karpovich, J. and Katramatos, D. *The Legion Resource Management System*. Proc. of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99), in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS '99), April, 1999. <http://www.cs.virginia.edu/~legion/papers/legionrm.pdf>
5. Cray, R. Document number in-2153 2/97. Cray Research, 1997.
6. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W. and Tuecke, S. *A Resource Management Architecture for Metacomputing Systems*. Proc. Of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pp. 62-82, 1998. <ftp://ftp.globus.org/pub/globus/papers/gram97.pdf>

7. Foster, I. *The Grid: A New Infrastructure for 21st Century Science*. Physics Today, Vol. 55, No. 2, pp. 42-47, February, 2002. <http://crystal.uta.edu/~levine/class/spring2003/grid/Foster_phyiscs_today.pdf>
8. Globus Dynamically-Updated Request Online Coallocator (DUROC) v0.8, Globus Alliance. http://www.globus.org/duroc/frames.html>
9. Henderson, R. and Tweten, D. *Portable Batch System: External reference specification*. Ames Research Center, 1996.
10. IBM. *IBM Load Leveler: User's Guide*. Document number SH26-7226_00, IBM Corporation. 1993.
11. Livny, M., Tannenbaum, T. and Thain, D. *Condor and the Grid*. in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003. <http://media.wiley.com/product_data/excerpt/90/04708531/0470853190.pdf>
12. Marsan, C., *Grid Vendors Target Corporate Applications*, Network World Fusion, January 2003.
13. Open Grid Services Infrastructure V1.0: Primer. OGSI Working Group, March 12th, 2004. <https://forge.gridforum.org/projects/ogsi-wg/docman/>.
14. Sandia National Laboratories, *JESS, the Java Expert System Shell*, http://herzberg.ca.sandia.gov/jess.
15. Thomas, R., *IBM and Schwab Tag Team on Linux Grid Computing*, Linux Electrons, January 2004.
16. Zhou, S., LSF*: Load Sharing in Large-Scale Heterogeneous Distributed Systems*. Workshop on Cluster Computing, 1992.