InteGrade[*]: object-oriented Grid middleware leveraging idle computing power of desktop machines

Andrei Goldchleger[†]
Fabio Kon
Alfredo Goldman
Marcelo Finger
Germano Capistrano Bezerra
{andgold,kon,gold,mfinger,germanob}@ime.usp.br

Department of Computer Science – University of São Paulo
Rua do Matão, 1010. 05508-090, São Paulo – SP, Brazil

SUMMARY

Grid computing technology improves the computing experiences at organizations by effectively integrating distributed computing resources. However, just a small fraction of currently available Grid infrastructures focuses on reutilization of existing commodity computing resources. This paper introduces *InteGrade*, a novel object-oriented middleware Grid infrastructure that focuses on leveraging the idle computing power of shared desktop machines. Its features include support for a wide range of parallel applications and mechanisms to assure that the owners of shared resources do not perceive any loss in the quality of service. A prototype implementation is under construction and the current version is available for download.

KEY WORDS: grid computing; distributed object systems; CORBA; usage pattern analysis

[*]Project Website: http://gsd.ime.usp.br/integrade
[†]Andrei Goldchleger is partially supported by a graduate fellowship from CAPES, Brazil
Contract/grant sponsor: CNPq, Brazil; contract/grant number: 55.2028/02-9

1. Introduction

In recent years we have witnessed an impressive growth in the use of computers in various fields of research, including physics, chemistry, biology, and economics. Even corporations rely on the intensive use of computing power to solve problems such as financial market simulations and studies for accurate oil well drilling. The movie industry makes intensive use of computers to render movies using an increasing number of special effects.

The recent introduction of clusters of commodity computers brought down the costs of the hardware needed to perform intensive computations. However, this solution has major drawbacks. First, traditional clusters are composed of dedicated machines. Even when no computation is being carried out, machines remain idle, normally inaccessible to other users. Second, the whole cluster infrastructure demands a lot of physical space, a temperature controlled environment, and measures to deal with the noise produced by cluster nodes. These problems may seem negligible when we consider a 16 node cluster, but when the cluster grows in size, they have to be considered carefully.

Meanwhile, when considering the computing resources available at corporations and universities, one sees that they typically have hundreds or thousands of desktop machines, which are used by workers as their personal workstations or by students in instructional and research laboratories. When analyzing the usage of each of these machines one concludes that they sit idle for a significant amount of time. Even when the computer is in use, it normally has a large portion of idle resources. When we consider the night period, we conclude that most of the times they are not used at all. This situation lives in contradiction with the large demand for computational resources in certain areas. The need for and waste of resources often coexist in the same institution. The problem is that organizations lack a software infrastructure to allow the efficient use of these idle resources.

This paper introduces *InteGrade*, a novel Grid middleware architecture to solve the contradiction mentioned above. The architecture enables a wide range of parallel applications to execute in a distributed environment, benefiting from the power of the hardware already available in organizations. This is achieved by integrating user desktop machines and resources in shared laboratories in an intranet or wide-area Computational Grid [1].

Differently from other grid approaches, InteGrade is based on state-of-the-art middleware technology, such as the CORBA [2] industry standard for distributed object systems. This allows us to leverage existing services, such as Naming [3], Trading [4], Transactions [5], and Persistence [6], shortening development and maintenance time. InteGrade services are exported as CORBA IDL interfaces being accessible from a large variety of programming languages and operating systems.

An important requirement for InteGrade is that users who decide to share their machines with the Grid shall not perceive any drop in the quality of service provided by their applications. InteGrade's architecture was carefully designed with this requirement in mind. Thus, client machines use a very lightweight CORBA implementation and the access to its hardware resources is carefully controlled by a user-level scheduler.

Although many applications can benefit from this environment, it is clear that parallel applications will benefit the most. Usually they have to be executed on dedicated resources, such as Beowulf clusters [7], but InteGrade allows them to execute over many shared resources, bringing the power of parallel computing to institutions that cannot afford a multi-node dedicated cluster. Dedicated resources can also be part of InteGrade grids, either remaining dedicated or converted to workstations and shared in the grid.

In such a dynamic environment, it is difficult to schedule the execution of applications, since an idle resource may become busy again without further notice. To minimize this problem, InteGrade's architecture includes a component for collecting and analyzing usage patterns, a mechanism that, based on usage information and statistics, can determine the probabilities of an idle node to become busy again. When tuned properly, this mechanism can help schedulers to foresee if an idle machine will stay idle for a significant amount of time or if it is going to be busy again in a few seconds.

The InteGrade project is a multi-university, 3-year initiative that started in the second semester of 2002. Section 2 discusses InteGrade's related work and Section 3 explains the major requirements and challenges the project will face. In Section 4 we present the proposed architecture, in Section 5 we detail the mechanism for collecting and analyzing usage patterns and in Section 6 we describe which parts of the architecture have already been implemented, which parts are still under development, and on what technologies the implementation is based. Finally, we present our conclusions in Section 7.

2. Related Work

In recent years, many Grid related projects surfaced, enabling new ways of resource sharing and integration.

The Globus project [8, 9] provides an infrastructure for the integration of several computers, spanning different geographical locations into a single grid system. It provides a toolkit that allows the construction of grid-enabled applications in an incremental fashion. While Globus focuses on building the software infrastructure for a wide range of machines, including high-end ones, InteGrade focuses on leveraging the idle processing power from commodity workstations, taking appropriate measures to ensure that their users do not feel any drop in the quality of service. Another difference is that InteGrade is being built using an object oriented architecture and the CORBA industry standard, which facilitates the interaction between the middleware system and applications through the use of well defined IDL interfaces. The use of CORBA also allows the middleware platform to be based on a distributed object model leveraging existing CORBA services, such as Trading, Naming, and others.

Legion [10, 11] provides a middleware infrastructure that enables applications to benefit from execution in a distributed and parallel environment. It provides its own specific runtime library and builds its services on top of a unified object model [12]. InteGrade differs from Legion in its use of CORBA instead of a proprietary distributed object model. InteGrade also has a deeper focus on idle resource management and commodity hardware.

Condor [13, 14] may be considered the pioneer of Grid systems. As InteGrade does, it focuses on harvesting idle computing power from workstations in order to perform useful computation, such as *High Throughput Computing* [15] tasks. It provides scheduling and monitoring to applications without the need of modifying them. It also provides checkpointing but this requires the application to be re-linked with a specific library. However, support for parallel applications is currently quite limited, since some computers in the system should be configured [16] as partially-reserved nodes, like nodes on a Beowulf cluster. The reservation might not be feasible, for example, if the node is used by an employee. Checkpointing of parallel applications is currently under development, and can address this issue. Differently from Condor, InteGrade is being built with parallel applications in mind from the beginning. We also plan on featuring mechanisms for collecting and analyzing usage patterns, which can be used to forecast the behavior of grid nodes to improve scheduling. Finally, InteGrade uses CORBA as its communication protocol rather than a specific one used in Condor.

The SETI@home project [17] built an infrastructure to solve a single problem, SETI (*Search for Extraterrestrial Intelligence*), and obtained support from thousands of users, leveraging the power of hundreds of thousands of commodity workstations throughout the world. It is accredited as the problem that has received the largest amount of computing time in history. Despite its tremendous success, this project has several limitations when compared to InteGrade, the most notable being the impossibility of solving different problems. Other limitations include the lack of support for parallel applications that demands communication between computing nodes, the necessary intervention of the client machines to specify when the application can run and the impossibility of using resources of a *partially* idle node.

BOINC (*Berkeley Open Infrastructure for Network Computing*) [18] is considered SETI@home's successor. It introduces many features that were missing in SETI@home, such as the possibility of using the grid to solve different problems, limited support for communication between parallel application nodes and checkpointing. Although development is in its beginning, the features planned for the full version lack usage pattern collection and analysis, one of InteGrade's features which will help to choose where applications should run. Differently from InteGrade, BOINC lacks general support for parallel applications, being more suitable for applications with negligible data dependencies between its nodes.

3. Architectural Requirements

One of InteGrade's most important requirements is to leverage idle resources of commodity computers. That requisite demands that the middleware have the means for determining the activity status in each of the grid nodes. To address that requisite, we need an information service that gathers all relevant information from each node. We also need the definition of what should be considered a candidate node to run a grid application. One could argue that a candidate node is any node that has any amount of free resources at a certain moment. This could be the basic definition used by the system, but to obtain the collaboration of a large number of users, we need to empower users with the means to determine when their machine resources will be exported to the grid and what portion of its resources can be used by grid applications. Thus, the system must provide a flexible and user-friendly way of letting resource providers share their machines as they want. On the other hand, we must also keep in mind that the vast majority of resource providers will not be knowledgeable users, so the system must provide sensible default values for its parameters to protect providers from degradation in the quality of service.

Another important requirement related to leveraging idle resources is ensuring that an application does make progress in its execution. We need the means to ensure that application execution evolves even in a dynamic environment in which nodes can turn from idle to busy without further notice. One solution is to use a checkpointing mechanism to ensure progress by periodically saving the application execution state. Since the grid can encompass different platforms and operating systems, this checkpointing must be machine and operating system independent to permit migration of computation across grid nodes. In order to minimize checkpointing situations, we make use of usage pattern collection and analysis. This procedure is based on information gathered by resource managers. Node usage information for short time intervals (e.g., 5 minutes) is grouped in larger intervals called periods. After that, the system shall apply clustering algorithms [19] to this data in order to extract behavioral categories. It is expected that these categories will map to common usage periods such as lunch-breaks, nights, holidays, working periods, and so on. From that mapping it will be possible to predict the time-span in which a machine will be idle. This is an evolutionary process: as data is being collected and analyzed new categories can appear, others can disappear.

Another major requirement is to provide support for a wide range of parallel applications, which is not the case in existing grid systems. Most grid initiatives provide support for parallel applications with little or no communication among application nodes. Some parallel applications demand more in terms of inter-process communication than others. The different demands in terms of communication suggest that different applications should be scheduled to different computers, respecting the differences regarding the network connectivity associated to each grid node. So, the distributed resource management service must also provide information on the kind of network connection available in each part of the grid. That kind of information will help schedulers to create virtual topologies based on the needs of parallel applications. With this kind of support, a grid user may, for example, submit the following request to InteGrade: *execute application $\mathcal{X}$ in two groups of 50 nodes, each group connected internally by a 100 Mbps network and the two groups connected by a 10 Mbps network; each node should have at least 16 MB of RAM and a CPU of at least 500 MIPS.*

The need to ensure application progress is even more challenging when parallel applications are considered. Measures that are adequate when sequential programs are considered are not directly applicable in parallel execution environments. Consider checkpointing, for example. If the system has to checkpoint a parallel application, what should it do with ongoing communications? How to determine that a process migrated to another machine, thus requiring all pending communications to be redirected? This kind of issue can render parallel checkpointing prohibitive, due to large overheads. Usage pattern analysis plays an important role since the scheduler can place parallel applications on idle nodes with lower probability of becoming busy before the computation is completed. Although usage patterns attenuate the problem, it does not solve it entirely since it can only provide a *hint* of what is going to happen and cannot guarantee the future availability of resources. We still need a model that saves the state of computation periodically, providing milestones that can be used to resume the application in case of crashes or when there is need for migration. To ensure that, InteGrade adopts BSP [20] as the model for parallel computation; imposing frequent synchronizations among application nodes.

Indeed, an important characteristic of the BSP model is the separation of the communication difficulties from the scheduling difficulties. For instance, given a parallel program with explicit parallelism (like a program written in MPI), its performance will vary depending on the allocation of the processes to the grid nodes, due to the heterogeneity of computers and network. Usually the programmer, or the scheduler, is responsible for finding an allocation which provides good enough performance (finding the optimal allocation is generally a hard problem). InteGrade's approach is to perform the allocation using information provided by the submission request (like the application $\mathcal{X}$ above), trying to provide fast links among the nodes that communicate the most. In addition, it is necessary to provide an efficient communication and synchronization mechanism through a complete exchange primitive. We are currently investigating how this primitive can be implemented using information about bandwidth and latency in the communication links (see [21, 22] for a complete description of our work on how to build primitives for complete exchange on homogeneous computers and on a cluster of clusters).

Finally, security must be considered throughout the entire architecture. The most important requirement is to ensure that users who decide to export their resources to the grid do not have their personal files and overall private information exposed or damaged in any way. To ensure that, we are investigating the use of bytecode and native code sandboxing [23] to protect from malicious code execution, as well as standard security mechanisms for authentication, access control, and cryptography.

4. Architecture

InteGrade grids are structured in clusters, each consisting of groups from one to approximately one hundred computers, which can be shared workstations or machines dedicated to the grid. Clusters are then arranged in a hierarchy, allowing a single InteGrade grid to encompass potentially millions of machines. The hierarchy can be arranged in any convenient manner. Figure 1 depicts the major types of components in a InteGrade cluster. The *Cluster Manager* node represents one or more nodes that are responsible for managing that cluster and communicating with managers in other clusters. A *User Node* is one belonging to a grid user who submits applications to the grid. A *Resource Provider Node*, typically a workstation, is one that exports part of its resources, making them available to grid users. A *Dedicated Node* is one reserved for grid computation. This kind of node is shown to stress that, if desired, InteGrade can also encompass dedicated resources. Note that these categories may overlap: for example, a node can be a *User Node* and a *Resource Provider Node* at the same time.

The *Local Resource Manager* (**LRM**) and the *Global Resource Manager* (**GRM**) cooperatively handle intra-cluster resource management. The LRM is executed in each cluster node, collecting information about the node status, such as memory, CPU, disk, and network utilization. LRMs send this information periodically to the GRM, which uses it for scheduling within the cluster. This process is called the *Information Update Protocol.*

The GRM and LRMs also collaborate in the *Resource Reservation and Execution Protocol*, which works as follows. When a grid user submits an application for execution, the GRM selects candidate nodes for execution, based on resource availability and application requirements. For

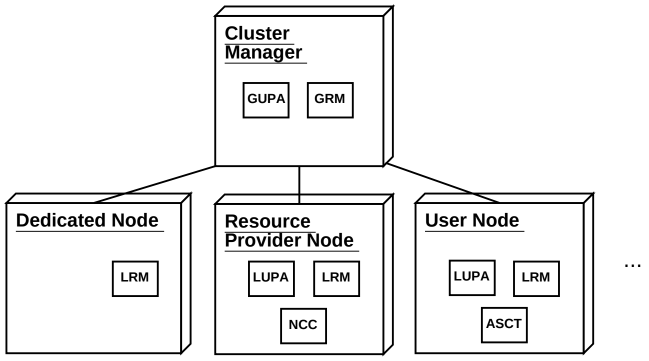

Figure 1. InteGrade's Intra-Cluster Architecture

that the GRM uses its local information about the cluster state as a hint for locating the best nodes to execute an application. After that, the GRM engages in a direct negotiation with the selected nodes to ensure that they actually have the sufficient resources to execute the application at that moment and, if possible, reserves the resources in the target nodes. In case the resources are not available in a certain node, the GRM selects another candidate node and repeats the process. The information, execution, and reservation protocols are based on previous work in the 2K Resource Management Subsystem [24]. A recent extension of this protocol [25] implemented by our group allows the GRM to engage in information updates, resource negotiation, and reservation across a collection of clusters connected through the Internet. Figure 2 depicts a possible cluster hierarchy. Resources in any of the clusters can be transparently accessed if needed.

Similarly to the LRM/GRM cooperation, the *Local Usage Pattern Analyzer* (**LUPA**) and the *Global Usage Pattern Analyzer* (**GUPA**) handle intra-cluster usage pattern collection and analysis. The LUPA executes in each cluster node that is a user workstation[*] and collects data about its user usage patterns. Based on long series of data, it derives usage patterns for that node throughout the week. Each node's usage pattern is periodically uploaded to the GUPA. This information is made available to the GRM, which can make better scheduling decisions due to the possibility of predicting a node's idle periods based on its usage patterns.

The *Node Control Center* **(NCC)** allows the owners of resource providing machines to set the conditions for resource sharing, if they so wish. Parameters such as periods in which they do not want their resources to be shared, the portion of resources that can be used by grid applications (e.g., 30% of the CPU and 50% of its physical memory), or definitions as to

[*]The LUPA is not executed in dedicated nodes that can only be used remotely.

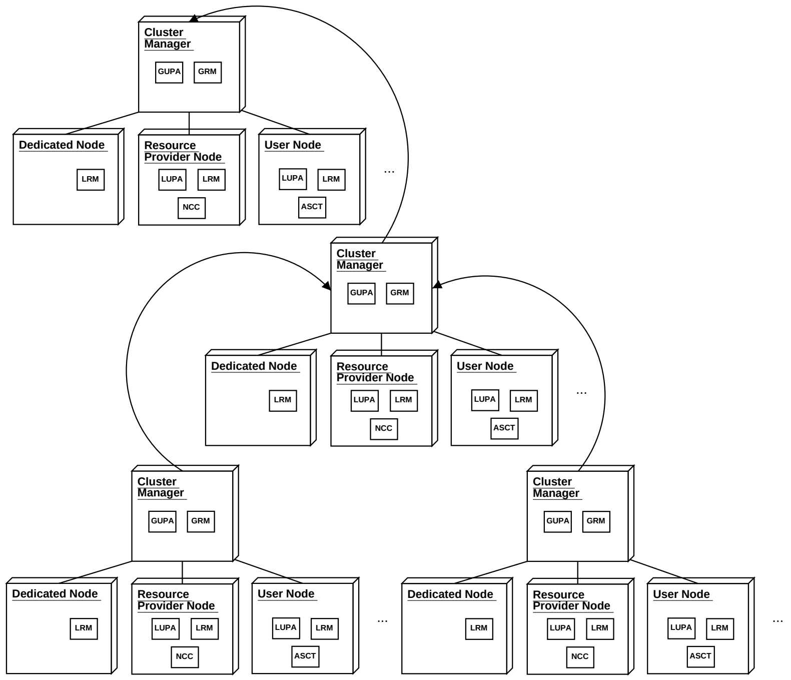

Figure 2. InteGrade's Inter-Cluster Architecture

when to consider their machine idle can be set using this tool. The *Dynamic Soft Real Time Scheduler*, part of DSRT [26], enforces the conditions of the resource owners.

The *Application Submission and Control Tool* **(ASCT)** allows InteGrade users to submit grid applications for execution. The user can specify execution prerequisites, such as hardware and software platforms, resource requirements such as minimum memory requirements, and preferences, like rather executing on a faster CPU than on a slower one. The user can also use the tool to monitor application progress.

5. Usage Pattern Analysis

In the InteGrade grid there are two kinds of tasks. *Local user tasks* are regular, centralized or distributed tasks performed by regular users on the network. *Grid user tasks* are distributed tasks performed on the grid using the idle computing power of grid machines. At all times, grid tasks should not interfere with local tasks, which implies that a grid task must be suspended when a local user task requires resources being used by the grid task.

The goal of usage pattern analysis is to detect which machine can receive a grid task at a certain moment, so as to minimize its completion time. The usage analysis is performed via a machine learning procedure.

Usage pattern analysis takes place in two levels in our system. At the local level, Local Usage Pattern Analyzer (LUPA) modules are installed in each grid machine. Its goal is to obtain information about how local users utilize basic machine resources across time, such as CPU, physical memory, virtual memory, disk space, etc. This information is collected as time series, and it is used in the prediction of how free or busy the machine will be at a certain time of the day, which will guide the decision of grid task allocation.

The *Global Usage Pattern Analyzer* (GUPA) receives a description of a grid task to be performed and indicates which grid machine under its jurisdiction, if any, is probable of performing that task within minimal time.

There is a conflict with respect to where should the data collected by the LUPA be analyzed to support the grid task allocation decision. There are two possible levels of analysis, namely:

- *GUPA level analysis.* In this case, all the data collected at the LUPA modules is forwarded to the GUPA, which proceeds to identify patterns of resource usage of all the machines under the its jurisdiction. The GUPA, then, has a global view of usage patterns of all machines in its domain, which it will use for the grid task allocation decision.
  This approach has the serious drawback of violating the *privacy* of local machine users. The fact that each machine user will have its usage pattern exposed to a global controller may serve as a negative incentive for users to allow their machines to take part of the grid. Privacy violation is a serious enough consideration on its own, but it also has consequences on the success of the grid deployment. If users do not allow their machines to be included in the grid, the effort of idle-time grid deployment will fail.
- *LUPA level analysis.* The LUPA collects and processes resource usage time series locally. At each moment in time, only the LUPA knows if the machine where the LUPA module is installed is probable to remain idle or not, so no privacy violation occurs. Upon receiving the description of a grid task, the GUPA has to poll the set of LUPAs to decide where the grid task must be sent to.
  This scheme also has the advantage of distributing the processing of usage time series, dividing this overhead over the grid.

For privacy and load balancing reasons, we have chosen to locate time series analysis at the local level.

Time Series Analysis

Time series analysis proceeds in a machine learning environment, and processing is done in two phases: the learning phase and the operational phase [27]. During the learning phase, the resource usage time series is collected for a period of time. We are collecting data from machines at intervals of 5 minutes, 24 hours a day (while the computer is on), over a period of several weeks.

The time series for each resource in analysis is then processed so as to determine the usual patterns of resource usage. This processing consists of a variation of unsupervised clustering of time series [28, 29]. The goal is to obtain a small, fixed number of clusters (or usage patterns). The variation of the clustering algorithm has the following aims:

- To reduce the number of clusters to a manageable size.
- To reduce the computational overhead of the clustering method to a tractable complexity. For this reason, we do not use standard clustering, but an approximate algorithm that, although not providing an optimal solution, is efficient and stays within a quadratic complexity [30].

Thus, the learning time produces a small set of usage patterns, in the form of *prototypical time series*.

During the operational phase, a decision on the availability of a grid machine will have to be reached. For that, a set of resource requirements is provided. The time series is still being recorded during the operational phase. The analyzer will look at the current time series and decide which prototypical time series is closer to the current one. With that information, it will classify the current time series, and is thus able to decide on the probable availability of the requested resources.

6. Implementation Status and Ongoing Work

Although we had implemented the intra- and inter-cluster protocols for information updates, resource reservation, and execution in the 2K [24, 25] system, we are currently re-implementing these protocols from scratch in the new InteGrade middleware platform. This was necessary due to InteGrade's strict requirements with respect to ensuring that the quality of service perceived by users sharing their machines with the grid would not be affected.

The new middleware platform is based on O^2 [31], a very small memory footprint CORBA-compatible ORB developed by InteGrade collaborators at PUC-Rio. O^2 is written in Lua [32] and can be accessed via a C/C++ API. In this new middleware platform, we have implemented the intra-cluster information protocol that allows LRMs to send node status to GRMs. We also implemented the intra-cluster execution protocol, that allows applications to be remotely executed in an InteGrade cluster. The LRM is currently implemented in C++ using O^2. The GRM, which runs on a server node, is implemented in Java on top of JacORB [33]. The GRM uses the JacORB Trader to store the information it receives from the LRMs.

Ongoing work includes the extention of the intra-cluster execution protocol, which will introduce extra flexibility in the system, such as the ability of executing parallel applications. We also started to collect information about node utilization in order to develop node usage patterns that will be used on LUPA and GUPA. Source-code and documentation for the latest version is available at the InteGrade Web site.

7. Conclusion

InteGrade will provide a middleware infrastructure to enable applications to leverage the idle computing power from commodity computers. Its key features are support for a wide range of parallel applications, use of advanced object-oriented techniques on architectural design and development, and node usage pattern collection, analysis, and prediction. This infrastructure will unlock the power of distributed parallel computing in organizations that cannot afford to have dedicated resources. It has also a great potential for lowering the level of waste of computational resources in today's computing infrastructure.

REFERENCES

1. Foster I (ed.), Kesselman C (ed.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers: San Francisco, 1999.
2. OMG. *CORBA v3.0 Specification*. Object Management Group: Needham, MA, July 2002. OMG document 02-06-33.
3. OMG. *Naming Service Specification*, version 1.2, September 2002. OMG document formal/02-09-02.
4. OMG. *Trading Object Service Specification*, version 1.0, June 2000. OMG document formal/00-06-27.
5. OMG. *Transaction Service Specification*, version 1.3, August 2002. OMG document formal/02-08-07.
6. OMG. *Persistent State Service Specification*, version 2.0, September 2002. OMG document formal/02-09-06.
7. Beowulf website.
   `http://www.beowulf.org` [1 August 2003].
8. Globus website.
   `http://www.globus.org` [1 August 2003].
9. Foster I, Kesselman C. Globus: a metacomputing infrastructure toolkit. *International Journal of Supercomputing Applications* 1997; **2**(11):115–128.
10. Legion website.
    `http://www.cs.virginia.edu/~legion` [1 August 2003].
11. Grimshaw AS, Wulf WmA. Legion – a view from 50,000 feet. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society Press: Los Alamitos, California, August 1996.
12. Lewis MJ, Grimshaw A. The core Legion object model. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society Press: Los Alamitos, California, August 1996.
13. Condor website.
    `http://www.cs.wisc.edu/condor` [1 August 2003].
14. Litzkow M, Livny M, Mutka M. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*. June 1988; 104–111.
15. Livny M, Basney J, Raman R, Tannenbaum T. Mechanisms for High Throughput Computing. *SPEEDUP Journal* June 1997; **11**(1).
16. Wright D. Cheap cycles from the desktop to the dedicated cluster: combining opportunistic and dedicated scheduling with Condor. In *Proceedings of the Linux Clusters: The HPC Revolution conference*. Champaign - Urbana, IL, June 2001.
17. SETI@home website.
    `http://setiathome.ssl.berkeley.edu` [1 August 2003].
18. BOINC website.
    `http://boinc.berkeley.edu` [1 August 2003].
19. Johnson RA, Wichern D. *Applied Multivariate Statistical Analysis*. Prentice-Hall, 1983.
20. Valiant LG. A bridging model for parallel computation. *Communications of the ACM* 1990; **33**(8):103–111.
21. Goldman A, Peters J, Trystram D. Exchanging messages of different sizes. Technical report, School of Computing Science, Simon Fraser University. September 2002.
22. Goldman A. Scalable algorithms for complete exchange on multi-cluster networks. In *2nd CCGRID*. May 2002; 286–287.
23. Goldberg I, Wagner D, Thomas R, Brewer EA. A secure environment for untrusted helper applications: confining the wily hacker. In *Proceedings of the USENIX Security Symposium*. July 1996.
24. Kon F, Yamane T, Hess C, Campbell R, Mickunas MD. Dynamic resource management and automatic configuration of distributed component systems. In *Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'2001)*. San Antonio, Texas, February 2001.

25. Marques JR, Kon F. Distributed resource management in large-scale systems *(in Portuguese)*. In *Proceedings of the 20th Brazilian Symposium on Computer Networks.* Búzios, Brazil, May 2002; 800–813.
26. Nahrstedt K, Chu HH, Narayan S. QoS-aware resource management for distributed multimedia applications. *Journal of High-Speed Networking, Special Issue on Multimedia Networking* 1998; **7**(3,4):227–255.
27. Mitchell TM. *Machine Learning.* Computer Science Series. McGraw-Hill, 1997.
28. Arabie P (ed.), Hubert LJ (ed.), Soete GD (ed.). *Clustering and Classification.* World Scientific, 1996.
29. Ryzin JV (ed.). *Classification and Clustering.* Academic Press, 1977.
30. Vaziran V. *Approximation Algorithms.* Springer Verlag, 2001.
31. O^2 website.
    `http://www.tecgraf.puc-rio.br/luaorb/o2` [1 August 2003].
32. Ierusalimschy R, Figueiredo LHd, Filho WC. Lua-an extensible extension language. *Software—Practice and Experience* 1996; **26**(6):635–652.
33. JacORB website.
    `http://www.jacorb.org` [1 August 2003].