Hierarchical Resource Management and Application Control in Grid Environments

Patrícia Kayser Vargas [1,2]
Inês de Castro Dutra [1]
Cláudio F. R. Geyer [3]

[1] COPPE - Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Caixa Postal 68511 – CEP 21949-972 – Rio de Janeiro, RJ, Brazil
{kayser,ines}@cos.ufrj.br
[2] Curso de Ciência da Computação
Centro Universitário La Salle
rua Victor Barreto, 2288 – CEP 92010-000 – Canoas, RS, Brazil
[3] Instituto de Informática
Universidade Federal do Rio de Grande Sul
Caixa Postal 15064 – CEP 91501-970 – Porto Alegre, RS, Brazil
geyer@inf.ufrgs.br

Abstract—

Several works on grid computing have been proposed in the last years. However, most work presented in the literature, including available software, can not deal properly with several issues such as loss of data, resubmission of faulty jobs, network control, data locality, unnecessary migration of transient data files, overload of submit machines, among others. This paper presents an ongoing work that deals with these limitations, focusing on applications that spread a very large number of tasks across the grid network. The central idea of our work is to have a *hierarchical* management system. A hierarchical management mechanism can control the execution of a huge number of distributed tasks preserving data locality while alleviating the load of the submit machine.

I. Introduction

Many applications have a high demand of computational resources such as CPU cycles and/or data storage. For instance, research in high energy physics (HEP) and DNA sequencing usually requires processing of large amounts of data using processing intensive algorithms.

The Compact Muon Solenoid project (CMS) estimates that 12-14 PetaBytes of data will be generated each year [28]. Dutra *et al* [11] reports experiments of inductive logic programming that generated over 40 thousand jobs that required many resources in order to terminate in a feasible time. These non trivial applications need a powerful distributed execution environment with many resources that currently are only available across different network sites.

The term *grid computing* [17, 15] was coined in the mid-1990s to denote a distributed computing infrastructure for scientific and engineering applications. A grid computing environment supports the sharing and coordinated use of heterogeneous and geographically distributed resources. These resources are made available transparently to the application independently of its location as if they belong to a unique and powerful computer. These resources can be CPUs, storage systems or network interconnection.

Several works on grid computing have been proposed in the last years. However, most work presented in the literature, including available software, cannot deal properly with several issues such as loss of data, resubmission of faulty jobs, network control, data locality, unnecessary migration of transient data files, overload of submit machines, among others. This paper presents an ongoing work that deals with these limitations, focusing on applications that spread a very large number of tasks across the grid network. The central idea of our work is to have a *hierarchical* management system. A hierarchical management mechanism can control the execution of a huge number of distributed tasks preserving data locality while alleviating the load of the submit machine.

The remaining of this text is organized as follows. Section II presents some of the several works on grid computing that have been proposed in the last years. Section III discusses some of the problems that need to be solved to satisfy user needs, which are the motivation to our model. We present and analyze our proposal to deal with some of these open problems in Section IV and presents some implementation issues in Section V. Finally, Section VI concludes this text with our final remarks and future work.

II. Grid Computing Systems

An increasing number of research groups have been working in the field of network wide-area distributed computing. They have been implementing middleware, libraries, and tools that allow cooperative use of geographically distributed resources. These initiatives have been known by several names [1, 7] such as metacomputing, global computing, and more recently grid computing.

Roure et al [7] identifies three generations in the evolution of grid systems. The *first generation* includes the forerunners of grid computing as we recognize it today and were projects to connect supercomputing sites. At the time this approach was known as metacomputing. The early to mid 1990s mark the emergence of the early metacomputing or grid environment.

Two representative projects in the first generation were FAFNER and I-WAY. FAFNER (Factoring via Network-Enabled Recursion) [13, 12] was created through a consortium to make RSA130 factorization using a numerical technique called Number Field Sieve. I-WAY (*The Information Wide Area Year*) [9] was an experimental high performance network that connected several high performance computers spread over seventeen universities and research centers using mainly ATM technology. These projects differ in some ways: (a) FAFNER was concerned with one specific application while I-WAY could execute different applications, mainly high performance applications; (b) FAFNER could use almost any kind of machine while I-WAY assumed high-performance computers with a high bandwidth and low latency network. Nevertheless, both had to overcome a number of similar obstacles, including communications, resource management, and the manipulation of remote data, to be able to work efficiently and effectively. Both projects are also pioneers on grid computing systems and helped to develop several second generation projects. FAFNER was the precursor of projects such as SETI@home (*The Search for Extraterrestrial Intelligence at Home*) [14] and Distributed.Net [10]. I-WAY was the predecessor of the Globus [30] and the Legion [21] projects.

The *second generation* projects have a focus on middleware to support large scale data and computation. The two most representative projects are Legion and Globus.

The Legion object oriented system [21, 5] was developed at the University of Virginia and it is now a commercial product of Avaki. In Legion, active objects communicate via remote method invocation. Some system responsibilities are delegated to the user level, as for example, Legion classes create and locate their objects as well are responsible for selecting the appropriate security mechanisms and the objects allocation policy.

The Globus Project has been developed by the Argonne National Laboratory, University of Southern California's Information Sciences Institute, and University of Chicago. The most important result of the Globus Project is the Globus Toolkit [32]. The Globus Toolkit [30] (GT) is an open source software. The GT version 2 can be classified as a second generation system since it is mainly a set of components that compose a middleware.

Finally, the *third generation* is the current generation where the emphasis shifts to distributed global collaboration, a service oriented approach, and information layer issues.

In the context of the third generation is the *Open Grid Services Architecture* (OGSA) [33] proposal that aims to define a new common and standard architecture for grid-based applications. The OGSI is a formal and technical specification of the concepts described in OGSA, including Grid Services. The version 3 of the Globus Toolkit (GT3) has a new philosophy of Grid services and implements the *Open Grid Service Infrastructure* (OGSI) [16].

Similar to the GT3 philosophy is the Semantic Grid proposal [8]. This architecture adopts a service-oriented perspective in which distinct stakeholders in the scientific process, represented as software agents, provide services to one another, under various service level agreements, in various forms of marketplace.

Another current classification of grid computing systems is computational and data grid [27]. The *Computational Grid* focuses on reducing execution time of applications that require a great number of computer processing cycles. The *Data Grid* provides the way to solve large scale data management problems. Data intensive applications such as High Energy Physics and Bioinformatics require both Computational and Data Grid features.

III. Motivation

Most software developed for grid environments solve several important issues, but lack features that deal with data loss, resubmission of faulty jobs, network control, data locality, unnecessary migration of transient data files, and overload of submit machines.

Next, we discuss each one of these issues.

A. Data Loss

Most available software can not handle network traffic properly. For example, one of the softwares that we had experience with, the Condor resource management [31] can either loose jobs that were in the job queue, or generate corrupt data files, because of lack of network flow control. The user is responsible to manually control the number of jobs that will be simultaneously submitted in order to avoid network congestion. As a consequence of the little attention given to flow control and data management data loss can occur due to overflow when too much traffic is generated on data and code transfers. Some experiments reported on [11] illustrate this problem. From 45,000 tasks launched, around 20% failed for several reasons, including data corrupted due to packet loss, and had to be re-submitted.

There are several works in the network research area that deal with network control flow [23, 19] and some of them could be adapted to grid applications needs.

MonaLISA [25], a software to monitor network and machine status, is a step towards the goal of achieving network flow in a grid environment and to avoid data and job loss.

B. Overload of Submit Machines

Usually, applications are launched in the user machine. Because most grid aware softwares create one connection or a new process to each launched job, the submit machine can become overloaded and have a very low response time, preventing any useful work to be done.

C. Resubmission of faulty jobs

In a grid environment we can assume that things can go wrong for several reasons. The source of the failure can be malfunctioning software or hardware. Most current grid aware softwares can deal with hardware faults or inclusion of new hardware. However, they can not deal properly with all fault possibilities. Therefore, we need to guarantee that jobs not completed because of some failure need to be re-submitted. To the best of our knowledge, no current grid aware software can deal with this problem.

D. Data Locality

Usually, tasks launched by the user have some kind of dependence on each other. Systems like Chimera [18] and DAGMan [31] allow the user to specify dependencies among tasks. They build a task graph where the nodes represent tasks and edges represent dependencies through data files. In that case, data locality must be preserved in order to avoid unnecessary data transfer. Because the available systems do not deal with this issue, transient files can be unnecessarily circulating in the network.

IV. Model

The main ideas to solve some of these problems presented on Section III are the following:

- to control execution of a huge number of distributed tasks;
- to manage data placement;
- to preserve data locality and to take advantage of transient data;
- to control network flow to avoid congestion;
- to use a hierarchical organization;
- to have adaptive behavior, i.e., management decisions can change due to system dynamics.

These main ideas can be concretized through a hierarchical management system. We present in the next sections some details of this model, starting with our assumptions.

A. Assumptions

The main premises assumed to the environment where our system will run are the following:

Heterogeneous environment We suppose that machines could have different software and hardware configurations, and probably will have as one could expect in any grid environment. This heterogeneity must be taken into consideration in our scheduling decisions.

A huge number of tasks can be submitted This assumption is fundamental in the definition of several details in our model. By a huge number of tasks we mean applications that generates thousands of jobs. These kinds of applications could not be easily controlled by hand.

Huge files can be used in computation Huge file transfer could cause network congestion. So, some kind of action must be taken to control file transfer avoiding package lost and network saturation. Besides, it takes a considerable transfer time. Data locality and caching techniques could help minimizing performance losses due to data transfer latency.

Tasks do not communicate by message passing This is not a strong imposed restriction, i.e., we probably will treat parallel application with message passing in the future. We just decided to postpone the analysis of this kind of application to narrow our initial scope, simplifying our model conception. Besides, we have at this moment several applications to use in our experiments that do not use message passing.

Tasks can have dependencies with other tasks due to file sharing A *bag-of-tasks* is the simpler way the application can be organized: there is no dependencies between tasks, so they can be executed in any order. Some Monte Carlo simulations can be classified in this group. The grid system MyGrid [6] is an example of a system that deal properly with this kind of application. The task can also can be seen as a dependency *graph* due to file sharing. For example, if a task a produces an output file f_a that task b uses as its input file, then b must wait until task a finishes. In this example, a and b are nodes and there is an edge from a to b, therefore b can only be launched after a finishes its execution. This is a common assumption as presented in Condor's DAGMan [31] and Globus' Chimera [18].

Huge number of files can be manipulated by tasks Tasks can communicate and synchronize through files, so, each task usually will manipulate at least two files (one input and one output). Since we assume a huge number of tasks, a very high number of files must be managed. Efficient algorithms to keep data locality and to efficiently transfer files are crucial to the success of the model under these assumptions.

UNIX operating system this is not exactly a model restriction. It is just a matter of simplification in our initial phase of implementation and of requirement detection. This assumption is based in our own environment where most machines have some UNIX-flavor, mostly Linux.

B. The Hierarchical Management System Overview

Figure 1 represents a schematic view of an application submission in our environment. This schematized view is presented to help understanding the function of all our model components.

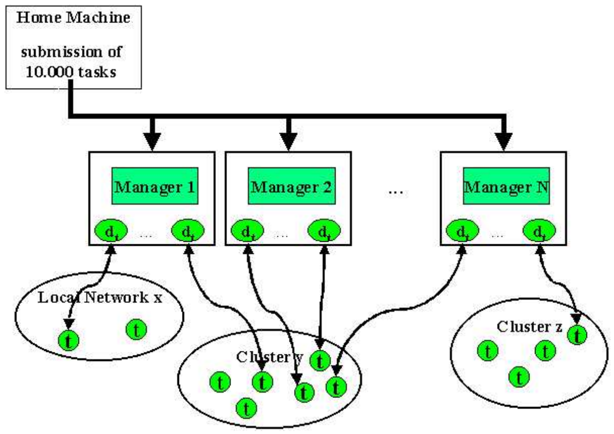

Figure 1. Hierarchical task dispatching example

First, the user submits in his home machine an application with, for example, 10 thousand tasks. If all the submission and the control were done in the home machine, probably this machine would stall and the user would not continue to work there. This problem is already known in the literature. For example, Condor [31] allows the user to specify a limit of jobs that can be submitted in a specific machine. This is not the best solution, since users must have some previous experience in job submission to infer the appropriated limit avoiding his/her machine stall and getting a good concurrency. So, in our model, this task submission process will be delegated to specialized managers.

Each manager, running in separate machines, can submit and take care of a set of tasks that present several task subgraphs. The Manager launches processes to accomplish the submission and the control. These processes are illustrated in Figure 1 as the d_t circles.

Then, each d_t process takes care of one or more task in a remote machine, illustrated in the figure as the t circles. The scheduler associated to the manager must choose the resources appropriated considering task specific characteristics such as memory usage, operating system version, and disk space.

The dynamics presented in this example becomes concrete in our model through the three components – `Master Submit`, `Submission Manager`, and `Task Manager` – that will be presented in the next subsection.

C. Model Components

Submitting a huge amount of tasks under the control of our system can only be done through a *Submit Machine*. A submit machine is a machine that has the `Master Submit` component installed. The `Master Submit` is in charge of:

- receiving an user input file describing the tasks to be executed. We are formalizing our description language trying to get the best ideas from our experience with other languages such as VDL [18], DAGMan [31], and ClassAds [29];
- partition the tasks in groups and send each group to a different `Submission Manager`. Ideally, each `Submission Manager` receives an independent group, i.e., any task depends on external task results. However, many applications could not be solved with this restriction. Then, when partitioning, the `Submit Machine` selects groups minimizing communication and distributes according to the `Submission Manager` capacity;
- show in a user friendly way the status and monitoring information received from all `Submission Managers`.

Note that the `Master Submit` does not have information about resources available in the system. It only keeps track of the `Submission Managers` status to avoid communicate with a failure node or with an overloaded one.

The `Submission Manager` main functions are :

- to do graph partitioning using the scheduling by edge reversal technique (SER) [20, 2]. SER is a scheduling mechanism based on the manipulation of acyclic orientations of a connected graph. We use this technique to get subgraph of related processes;
- to create daemons called `Task Manager` to control real task execution. Each daemon keeps control of a subgraph of tasks defined by the partitioning;
- to keep information about computational resources;
- to communicate periodically with the `Submission Manager`. The idea is to work using the lease concept, in a similar way that Sun's JINI model [24]. A lease is a grant of guaranteed access over a time period;
- to supply monitoring and status information useful to the user. It stores in log files the information in a synthetic way. These log files can be consulted by the user but are not intended to be human readable. These information are sent to the `Master Submit` that has the responsibility to present data to the user. This periodic information flow is also used to detect failures.

The `Task Manager` is responsible for communicating with remote machines and launching remote jobs. Its main functions are :

- to communicate with the remote scheduler to negotiate remote execution task;
- to run tasks according to task graph;
- to allocate tasks keeping data locality;

- to keep information about task graph evolution;
- to control data transfer avoiding data loss and congestion.

The assumption of a huge number of tasks has important consequences to scheduling policy design. We cannot just submit jobs without controlling system parameters and flow control as some systems. For example, MyGrid [6] considers that making a fast scheduling decision is more important. It is true for many applications, but for our target applications we need to keep track of system information. For example, if the application take several days to finish, two seconds to find a suitable cluster is not a problem. The key point here is adaptation as we can see in the following paragraphs.

Figure 2 illustrates the three main components of our model and their relationship.

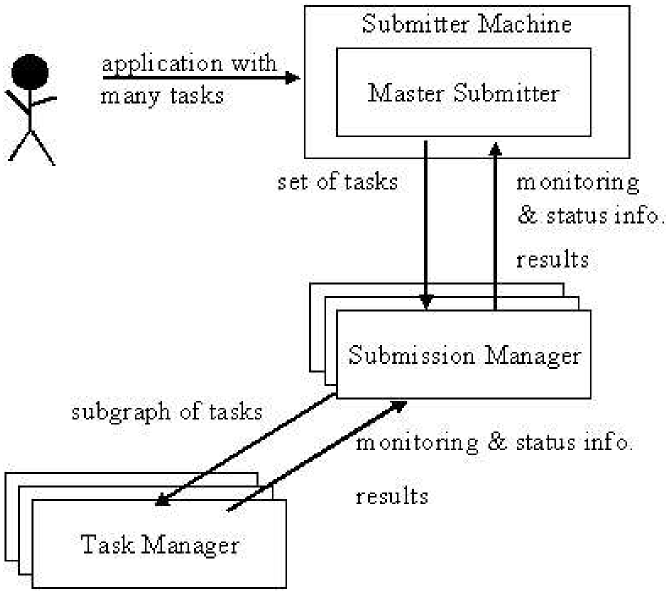

Figure 2. Hierarchical task management main components

When the user submits its application in the Submit Machine, the `Master Submit` can already be active or can be started due to the current request. When a `Master Submit` becomes active, it broadcasts a message to its local network. All `Submission Managers` replies to this message to inform their location and status. When an application submission request arrives in the active `Master Submit`, it uses its local information and choose one or more `Submission Managers` to accomplish all the required tasks. Then, periodically the `Submission Managers` will communicate with the `Master Submit` to indicate the execution progress. This protocol allows online monitoring information to the user and also fault detection.

There is communication between the `Submission Managers`, since some task groups can have dependence. Therefore some synchronization points must be established. The `Master Submit` must send, included in the group task description, the identification of each manager that is related to the group.

Our `Submission Manager` has also a functionality similar to the expander daemon of Vadhiyar & Dongarra's metascheduler [34]. Task execution occurs through the instantiation of the `Task Managers`. Each `Task Manager` has the responsibility of executing a subgraph of tasks in a specific location. This decision is made considering information that may be out dated. Then, while the `Task Managers` execute and report progress, the `Submission Manager` can find a better alternative to execute the tasks.

D. Flow Control

In our context, flow control is the management of data flow between nodes in a network so that the data can be handled at an efficient rate. Too much data arriving before a device can handle it causes data overflow.

Our proposal is to detect the network characteristics and to send data according to its latency and bandwidth. The flow must adapt according to the current status: sending input or output files can be postponed until the network can handle it. It is important to remember that our distributed organization is suitable to make this kind of control. Data received from remote sites in more than one machine can minimize contention. Note that each component must keep track of this information, but all should send its data. For example, if a `Submission Manager` waits too much to send its data, it may run of disk space.

V. Implementation Issues

We are presenting an ongoing work. Our initial prototype has been implemented mainly in C language due to performance requirements. The Scheduling by Edge Reversal technique is already implemented and we have been doing some experiments in the context of partitioning of Constraint Satisfaction Problems.

In a first approach, we decided to perform several simulations of our modelling to make some implementation decisions. Simulation has the advantage of allowing the control of several parameters as for example number of nodes in a cluster, bandwidth, latency, and mean time between failures. Besides, experiments in a simulation environment can be reproduced. At this moment, we are designing our simulation model. We intend to use a simulation framework as for example GridSim [3], MicroGrid [22] or the Monarc 2 Simulator [26].

VI. Conclusion

This paper presents a general framework for grid environments, whose central idea is to have a hierarchical organization where load of the user machine (submit) is shared with other machines. Our proposal wants to take advantage of hierarchical structures, because this seems to be the most appropriate organization for grid environments. We discussed some problems that must be solved in grid environments such as data loss, automatic resubmission of faulty jobs, data locality, and overload of submit machines. The solution proposed focuses on these problems. As far as the authors know, this is the first proposal of a hierarchical management system for grid environments.

We have implemented and already made some experiments with the partitioning algorithms needed by the `Master Submit` and by the `Submission Manager` components. We are now designing our simulation model. With simulation, several parameters will be under control and we will try to get the best options to our system implementation.

As future work, we must finish the implementation of our model. We intend to test it using applications from engineering, through a collaboration with the Laboratório de Projeto de Circuitos (LPC) at UFRJ, and from high energy physics, through the HEPGrid collaboration. Another important work is to define the characteristics of grid applications communication, in a similar way that has already been done to TCP [4]. This characterization could be used to get a more effective flow control policy.

References

[1] M. Baker, R. Buyya, and D. Laforenza. The grid: International efforts in global computing. In *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2000)*, Rome, Italy, Jul. 31 – August 6 2000. Also available at `http://www.cs.mu.oz.au/~raj/papers/TheGrid.pdf`.

[2] V. C. Barbosa and E. Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Transactions on Programming Languages and Systems*, 11(4):562–584, Oct. 1989.

[3] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14(13–15), Nov.–Dec. 2002.

[4] R. Cáceres, P. B. Danzig, S. Jamin, and D. J. Mitzel. Characteristics of wide-area tcp/ip conversations. In *Proceedings of the conference on Communications architecture & protocols*, pages 101–112. ACM Press, 1991.

[5] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. Resource management in legion. In *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99), in conjunction with IPDPS '99*, Apr. 1999. Also available at `http://legion.virginia.edu/papers/legionrm.pdf`.

[6] W. Cirne and K. Marzullo. Opengrid: a user-centric approach for grid computing. In *Proceedings of the 13th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2001)*, Sep. 2001. Also available at `http://walfredo.dsc.ufpb.br/papers/open-grid-sbac-final.ps`.

[7] D. de Roure, M. A. Baker, N. R. Jennings, and N. R. Shadbolt. The evolution of the grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 65–100. Wiley & Sons, 2003. Also available at `http://www.semanticgrid.org/documents/evolution/evolution.pdf`.

[8] D. de Roure, N. R. Jennings, and N. R. Shadbolt. The semantic grid: A future e-science infrastructure. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 437–470. Wiley & Sons, 2003. Also available at `http://www.semanticgrid.org/documents/semgrid-journal/semgrid-journal.p%df`.

[9] T. A. DeFanti, I. Foster, M. E. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide-area visual supercomputing. *The International Journal of Supercomputer Applications and High Performance Computing*, 10(2/3):123–131, Summer/Fall 1996. Available at `citeseer.nj.nec.com/article/defanti96overview.html`.

[10] Distributed.Net. `http://www.distributed.net/`.

[11] I. C. Dutra, D. Page, V. Santos Costa, J. Shavlik, and M. Waddell. Toward management of embarrassingly parallel applica-

tions. In *(Europar 2003)*, Klagenfurt, Austria, Aug. 2003. to appear.

[12] Fafner overview. `http://www.npac.syr.edu/factoring/overview.html`.

[13] Rsa130: Getting started with fafner. `http://cs-www.bu.edu/cgi-bin/FAFNER/factor.pl`.

[14] SETI@home The Search for Extraterrestrial Intelligence at Home. `http://setiathome.ssl.berkeley.edu/`.

[15] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1990.

[16] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, Jun. 2002. Available at `http://www.globus.org/research/papers/ogsa.pdf`.

[17] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001. Also available at `http://www.globus.org/research/papers/anatomy.pdf`.

[18] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, Jul. 2002. Also available at http://www.globus.org/research/papers/VDS02.pdf.

[19] E. W. Fulp and D. S. Reeves. Distributed network flow control based on dynamic competive markets. In *Proceedings International Conference on Network Protocol (ICNP'98)*, Austin Texas, Oct. 13-16 1998. Available at `http://citeseer.nj.nec.com/fulp98distributed.html`.

[20] E. M. Gafni and D. P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 1(29):11–18, Jan. 1981.

[21] A. S. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Wide-area computing: Resource sharing on a large scale. *IEEE Computer*, pages 29–36, May 1999. Also available at `http://www.cs.cornell.edu/Courses/cs614/2003SP/papers/legion-ieeecomp.p%df`.

[22] X. Liu, H. Xia, and A. Chien. Network emulation tools for modeling grid behaviors. Submitted to The 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003). Available at `http://www-csag.ucsd.edu/papers/ccgrid2003-final.pdf`.

[23] L. Massoulié and J. Roberts. Bandwidth sharing: objectives and algorithms. *IEEE/ACM Transactions on Networking (TON)*, 10(3):320–328, 2002.

[24] Sun Microsystems. Jini architecture specification, Jun. 2003. Also available at `http://wwws.sun.com/software/jini/specs/jini2_0.pdf`.

[25] H. B. Newman, I C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu. Monalisa: A distributed monitoring service architecture. In *Computing in High Energy and Nuclear Physics (CHEP03)*, La Jolla, California, USA, Mar. 24-28 2003.

[26] The MONARC Project Models of Networked Analysis at Regional Centres for LHC Experiments. Distributed computing simulation. `http://monarc.web.cern.ch/MONARC/sim_tool/`.

[27] S.-M. Park and J.-H. Kim. Chameleon: A resource scheduler in a data grid environment. In *3st International Symposium on Cluster Computing and the Grid*, pages 258–, Tokyo, Japan, May 12 - 15 2003.

[28] The Compact Muon Solenoid (CMS) Project. `http://lcg.web.cern.ch/LCG/`.

[29] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, Chicago, USA, Jul. 28-31 1998. Also available at http://www.cs.wisc.edu/condor/doc/hpdc98.ps.

[30] T. Sandholm and J. Gawor. Globus toolkit 3 core – a grid service container framework. White paper. Available at `http://www-unix.globus.org/toolkit/3.0beta/ogsa/docs/gt3_core.pdf`.

[31] D. Thain, T. Tannenbau, and M. Livny. *Condor and the Grid*, chapter BERMAN, Fran et al (editors). Grid Computing: Making The Global Infrastructure a Reality. John Wiley, 2003. Available at `http://media.wiley.com/product_data/excerpt/90/04708531/0470853190.pdf`.

[32] The Globus Toolkit. `http://www.globus.org/toolkit/`.

[33] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, and P. Vanderbilt. Grid service specification, 2002. Available at `http://www.gridforum.org/ogsi-wg/drafts/GS_Spec_draft03_2002-07-17.pdf`.

[34] S. Vadhiyar and J. Dongarra. A metascheduler for the grid. In *11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, pages 343–354, Edinburgh, Scotland, Jul. 24 – 26 2002. Also available at `http://hipersoft.cs.rice.edu/grads/publications/hpdc_meta.pdf`.