EasyGrid: Towards a Framework for the Automatic Grid Enabling of MPI Applications

Cristina Boeres and Vinod E. F. Rebello*
Instituto de Computação, Universidade Federal Fluminense (UFF)
Niterói, RJ, Brazil
e-mail: {boeres,vinod}@ic.uff.br

Abstract

The *Computational Grid* appears to have the potential to become an important and powerful computing platform in both the scientific and commercial distributed computing communities for the execution of large-scale applications. One of the goals of the grid is to aggregate collections of shared, heterogeneous, and distributed resources to provide computational "power" to parallel applications. However, designing applications capable of exploiting this potential easily remains a challenge. This paper outlines the EasyGrid methodology for the efficient and robust execution of MPI programs across distributed computing clusters. The principal objective of this work is to identify the application-oriented middleware necessary for, as well as develop a framework to automatically generate, *system-aware* (grid-enabled) applications capable of executing in such dynamic, unstable, distributed environments.

1 Introduction

With the high costs of acquiring and maintaining traditional supercomputers, the use of networks of workstations or PCs to tackle computation intensive applications has rapidly increased [1]. The concept of parallel processing using local networks and clusters of processors (commonly known as *Cluster Computing* [1]) has recently been extended to computing across networks of geographically distributed resources (*Grid Computing* [2]). Both cluster and grid computing are viewed as a low cost means of harnessing supercomputing-like performance due to their flexibility and scalability in comparison to traditional parallel machines. This cost benefit also means that a much larger community of users than before now have the opportunity to solve parallel computational problems on a widely accessible platform.

Grid computing adopted both its name and concept from the electrical power grid to capture the notion or **vision** of delivering computational performance efficiently, at a reasonable cost, according to demand, to anyone that needs it [2]. The realization of this vision and thus the success of the grid computing revolution will depend on the research community's ability to implement either an execution environment from which traditional applications draw performance with ease, or a programming environment to aid application writers develop new applications capable of executing efficiently on the grid.

This paper focuses on Computational Grids (rather than Service Grids or Data Grids [2]) for the execution of applications which require large amounts of distributed or parallel computation. One of the popular advantages of such grids is that they make computing power available to users who themselves have insufficient resources locally to execute their applications. But despite this, relatively few grid applications exist to exploit this new computing environment. To date, the majority of grid applications have been written by grid specialists and not "typical" scientists or engineers.

Given the diverse range of resources and dynamic behavior encountered in grid environments, developing grid enabled applications present significant challenges. Grid applications need to be designed to exploit the heterogeneous capacities of the resources available and overcome the negative effects caused by fluctuations in both performance and availability of these shared resources. If writing efficient programs for stable, dedicated parallel machines is already difficult, for the grid the problem is even harder. This factor alone is sufficient to inhibit the wide acceptance of grid computing.

In an attempt to bridge the gap between grid infrastructure and applications, much research is being focused at the *middleware* level, e.g. the Globus Project [3]. Many of these middleware services are relatively new and are constantly evolving, thus the services available may vary from site to site. In order to take advantage of these services, programmers require not only to be acquainted with the problem to be solved but also to have an intimate understanding of the functionality available from the middleware installed on the resources upon which the application will execute. Even if a fully de-

*The authors are funded by research grants from the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil.

veloped middleware were available, it is questionable if typical programmers would be knowledgeable enough to write grid applications capable of using these services. While a grid application is expected to execute in different environments, the application programmer cannot be expected to develop and maintain different versions for each of them.

If grid computing is to fulfill the vision of being accessible to ordinary programmers, developing grid applications must be made *easy*. The challenge is to get applications to execute efficiently and robustly in grid environments without placing this burdening the programmer or the user. This work-in-progress paper presents a brief overview of a framework for the automatic grid enabling of MPI applications. Named EasyGrid, the framework aims to address issues of application portability, execution efficiency and robustness to resource failures. Section 2 outlines the principal motivating factors behind the framework and other related work. Section 3 presents the EasyGrid methodology and describes the framework. Finally, Section 4 draws some conclusions and outlines ongoing work.

2 Motivation and Related Work

The acceptance of innovative techniques for improving solutions to computational problems is often hindered by the fact that, in order to benefit, applications are required to be rewritten. Ideally, the same application should be capable of executing efficiently equally on conventional parallel machines (for example a cluster of PCs or a supercomputer) and on a computational grid without the need for the programmer to modify the application.

New environments or frameworks need to be created to allow application developers take advantage of new grid capabilities easily. Realizing this objective requires two undertakings [4]: designing an interface which insulate programmers and users from the underlying complexity of grid environments without sacrificing the application's performance; and providing an execution environment that automatically and efficiently adapts the application to the dynamically changing resources of a grid. The current approach to application development focuses on the first of these i.e. implementing grid-enabling computational frameworks (e.g. AppLeS [5], Cactus [6], the GrAD Project [4] and GridLab [7]). The principal objective of these frameworks is to provide appropriate abstractions to grid services and a means for these to be accessed easily from within an application. Unfortunately, to execute on a grid, applications still need to be modified to interact with the appropriate service components available within the framework. Thus, require the application developer to be "grid-wise".

Cost effective computing depends on efficient scheduling schemes to harness the computing power available. However the decomposition of applications, resource management and scheduling in grid systems are complex activities [5]. Without any means to alleviate the burden on the programmer to solve these problems, truly realizing the grid vision will be difficult. For Grid computing to be successful, it has to be made *easier* and *efficient*. The EasyGrid Project aims to address this issue. The objective is to the secure the realization of the grid vision by making parallel computing on the grid accessible to "traditional" programmers. EasyGrid is a framework for the automatic transformation of MPI parallel applications into system-aware ones, and a methodology to investigate application-based middleware for Computational Grids. System-aware applications are adaptive, robust to resource failure (fault tolerant), self-scheduling programs capable of executing efficiently in shared, dynamic, unstable distributed environments like the Grid. In other words, they are applications which can react to changes in their executing environment. In the future, this can be extended to designing parallel applications which, depending on the resources available, dynamically alter the algorithm employed to solve the given problem.

3 The EasyGrid Methodology

While both the heterogeneous and dynamic nature of grids limit the applicability of tools already available for parallel systems, they also make developing system software and tools a necessity and a challenge. High performance for general MIMD programs requires effective scheduling and fault tolerance mechanisms. These considerations suggest that although grids may be constructed from existing parallel and distributed technologies, significant advances in mechanisms, techniques and tools will be of the utmost importance [2]. The EasyGrid methodology aims to allow programmers to concentrate on how to exploit parallelism to resolve the problem (within the chosen programming model) by using the *EasyGrid framework* to generate automatically a system-aware application capable of utilizing the grid resources available to the user in the most appropriate manner. The methodology adopts a middleware approach directed at the user or, more specifically, the application. The framework will be used to validate this approach and, in particular, to study both the static and dynamic scheduling problem and the integration of fault tolerance and scheduling strategies for (portable) system-aware applications on Computational Grids.

When talking about grids, most people adopt a *system-centric* viewpoint (see Figure 1), i.e. the grid consists of a set of resources upon which a pre-installed middleware system acts as a *Resource Management System* (RMS) to control the execution of grid applications. The RMS aims to achieve an efficient utilization of the resources

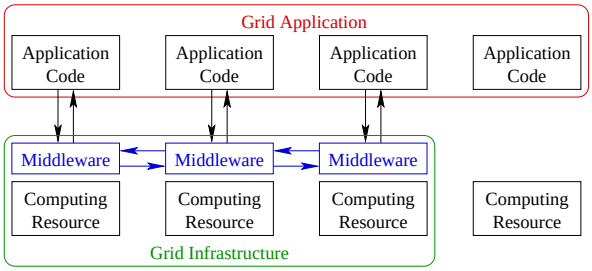

Figure 1: A simplified view of a system-centric Grid.

by monitoring and analyzing system specific information e.g. processor workload, network throughput etc., (metrics which do not distinguish individual application characteristics). However, to obtain good (parallel program) performance from the grid, it may not be sufficient for the RMS to simply *react* to the current behavior of the executing applications. Given the scale of grid systems, being able to predict the behavior of each application will play an important role in improving efficiency [8]. Employing a system-oriented RMS to adjust resource workload may not be the ideal approach. A better alternative might be to have the applications themselves adjust their requirements.

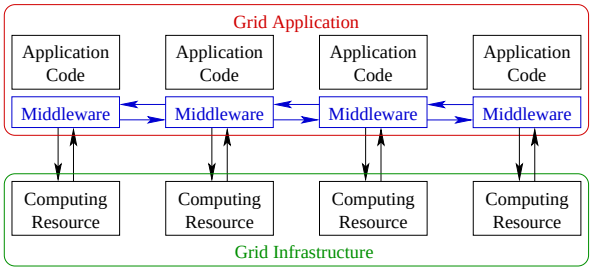

Figure 2: Simplified view of an application-centric Grid.

The EasyGrid methodology's view is *application-centric* (see Figure 2), i.e. the middleware provides services oriented towards the individual application. Efficient resource utilization is achieved in a distributed manner using an *Application Management System* (AMS) within each application. Each AMS has the objective of matching its own application requirements with the resources available. In other words, each application appears to have exclusive access to a virtual grid. Since the associated resources are being shared, each virtual grid actually runs at some fraction of its potential. From a system-centric viewpoint, the middleware is associated with the grid infrastructure whereas from an application-centric viewpoint the middleware is associated with each application. In the case of the former, a RMS tries to decide which applications may or may not use a given resource at particular moment. In the latter, each application decides dynamically which of the available resources to use, i.e. adjusts its execution to better exploit the current state of the grid environment.

Due to the dynamic nature of Grids, the set of usable resources will likely change from run to run or even during an application's execution. Portability (system-based middleware independence) allows an application to avoid being limited to executing on resources with a specific grid middleware (access and security issues aside), e.g. one resource in Figure 1 is unused since it lacks the appropriate middleware. Having access to a larger resource pool increases the chance for faster execution. Avoiding the need for the existence of a specific pre-installed middleware also facilitate the dynamic integration of various non-grid networks (e.g. university computing laboratories) into a single computing resource.

3.1 The EasyGrid Framework

This framework has the objective of being able to *automatically* convert traditional parallel programs into system-aware ones which execute efficiently on computational grids. In order to offer grid computing to a large group of potential users (Grid vision), we initially focus on parallel applications written with MPI due to its widespread use in parallel programming.

It is not to difficult to determine an efficient form of execution for coarse-grain, communication insensitive applications such as data mining, Monte Carlo simulations, parameter-space searches, etc. Typically, these applications have more than sufficient parallelism to hide the adverse effects of communication as well as the RMS processing costs. On the other hand, the performance of finer-grain (or relatively small, coarse-grain) applications is more susceptible to these costs. Good task- and communication-scheduling is crucial for the application to achieve acceptable performance. Our research focuses on adapting LogP scheduling algorithms based on task replication [9] as a technique to minimize the effects of relatively high communication costs between grid sites and offer some degree of tolerance in the face of resource failures. (The *LogP model* [10], a communication model which also considers the fact that networks have a limited communication capacity, has been shown to model accurately a variety of systems including grid-like wide area networks [11].)

Figure 3 presents the framework's compilation phase which generates an EasyGrid system-aware application from a user's MPI program. In the figure, functions are represented by rectangles with rounded corners and files by regular rectangles. In addition to the user's MPI program, a list of grid resources (Grid Access File) to which the users has access is also needed. The System Modeler creates a LogP architecture model by collecting information (processor speed and average load, LogP parameter

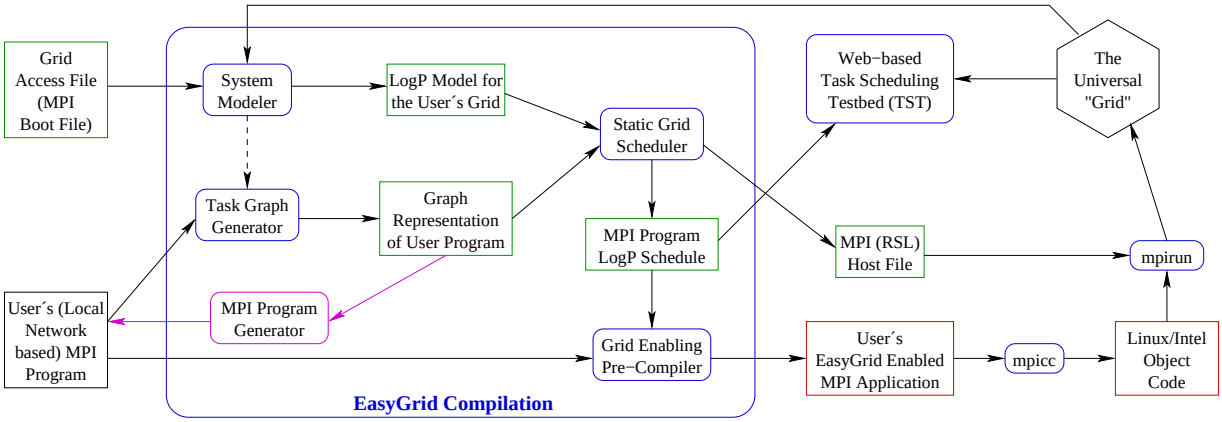

Figure 3: Generating an EasyGrid application.

values) for each of the user's resources (if online). Predicting networking behavior on the grid is an active research area and a number of grid monitoring tools and techniques for estimating network and resource behavior have been proposed [5, 8]. While questions remain over the accuracy of the information obtained, this work aims to investigate how this affects scheduling decisions.

Based on the characteristics of the grid resources and the application, an initial mapping or schedule is produced by the Static Grid Scheduler (i.e. a LogP task scheduling algorithm such as [9]) to guide the runtime execution. The MPI Host File identifies the subset of resources available to which MPI processes should be allocated. In task scheduling, a parallel application is often represented by a *directed acyclic graph* or *DAG* where nodes denote *tasks* (in this case, communication free portions of MPI processes) and whose edges denote data dependencies (MPI communications). While the Task Graph Generator can used to create a DAG representation of the user's program [12], the MPI Program Generator creates synthetic MPI programs from DAG representations. This facilitates the investigation of the effects of program structure and granularity (using scheduling benchmark DAGs) on the scheduling algorithms and grid execution. As well as an educational tool to compare and analyze schedules produced by Static Grid Scheduler implementations, the Task Scheduling Testbed can also monitor an EasyGrid Application's execution and compare the actual execution with the prediction. This is useful to help determine the importance of the System Modeller and the benefits of the initial static schedule.

The Grid Enabling Pre-compiler generates the EasyGrid MPI application by using the schedule to restructure the user's MPI program (this involves duplicating MPI processes and reassigning communications, if considered advantageous) and incorporating the appropriate application specific middleware (the AMS) both without altering any line of the user's code. While the user's application is transformed automatically, for debugging purposes there must be clear correspondence between the code written by the user and the code that executes on the grid. Therefore, the AMS MPI processes are effectively hidden from the user's code. The EasyGrid application with its AMS is distributed at three levels: a Home or Global Manager Node, responsible for the global execution of the program across different sites; Site Manager Nodes to control the execution at their respective sites, and; Local Processing Nodes to execute the user's code. Whether or not, Manager Nodes also execute user code is determined by the schedule.

Fundamental to the effective implementation of system-aware applications is determining what information must be monitored, collected and processed by the AMS. The foundation of the AMS is the application's "on-the-fly" self-monitoring system which provide information to other AMS functions (e.g. dynamic scheduling, fault tolerance). A side effect is the degree of intrusion suffered by the user program's execution. Currently, each local processing node monitors MPI communications in the user code (i.e. tracks the execution), passing on this information to the site manager. As well as passing the information from all of its processing nodes to the home node, the site manager's other AMS functions use the information to make site local decisions. An initial analysis based on synthetic MPI programs show that EasyGrid AMS monitoring incurs less than 5% intrusion (around 1% on average), using MPICH-G2 [3] from the Globus Toolkit to run on the GridRio Computational Grid [13].

The AMS's dynamic scheduler aims to fine tune the initial static schedule to improve the program's execution time. One mechanism being investigated is the use of scheduling windows (i.e. the period of time within which a task (process) may be executed without prolonging the estimated makespan of the application) identified by the static scheduler. If a task on a processing node starts ex-

ecuting outside of its scheduling window, the site manager's dynamic scheduler will decide if it is worthwhile either, moving tasks assigned to that node to another at the same site, or rescheduling *communication*. Due to task replication, there may not be a need to execute a delayed task since an existing copy elsewhere may be able to provide the necessary data sooner. Since every node has a copy of the MPI program, the dynamic scheduling function on each local processing node is responsible for enabling and disabling processes and updating a process communication table. (Note, for example, if a task with two successors is replicated, then each copy need only send one message. Since the user's code is not modified, this table identifies which message send commands are executed and should be inhibited.) The fault tolerance mechanism responds to resource failure in a similar fashion. Using monitoring information to detect that processes have (or will) not met their scheduling window, the site manager's fault tolerance mechanism asks the dynamic scheduler to reschedule the respective tasks.

4 Conclusions and On-going Work

Users frequently complain of the difficulty of achieving a reasonable fraction of the theoretical peak performance on the parallel systems they use. In fact, with computing systems continuously evolving and software becoming increasingly more complex, a growing number of problems related to performance make writing efficient applications a complex and often counter-intuitive task.

Finding a good balance between programmability and performance is one of the major challenges in realizing the Grid vision. But deciding on an appropriate programming model for the grid will take time. Since grid resources are available today, this work aims to make running parallel applications on the grid easier, rather than making parallel programming easier. The goal of the EasyGrid methodology is to allow programmers to focus solely on exposing the parallelism within the problem and rely on the EasyGrid Framework to automatically restructure the application to execute efficiently on the Grid resources available. The objective is to relieve the programmer of the complex task of grid enabling applications and thus implementing one program for a locally available computing platform and another for the grid.

In terms of portability and efficiency, the EasyGrid Framework adopts an application-centric middleware approach. The effectiveness of scheduling algorithms based on communication models which consider both hardware and software overheads and optimization techniques like task replication [9] are the focus of investigation. Many functions associated with grid infrastructure or middleware are essential for applications to be able to benefit from executing on the grid. However, up to now, little has been done to quantify the computational (and communication) overheads associated with executing middleware services or to evaluate how much these services affect the performance of applications. Our objective is to find the right computational balance between application and middleware workload and between system- and application-based middleware.

The mechanisms currently under investigation do not consider the possibility of new resources becoming available during the application's execution. Future work will look to incorporate other middleware functions, such as resource discovery. Another issue is how other EasyGrid and native applications which share the computing resources influence this EasyGrid application's execution. There is scope for future work to investigate the possibility of independent EasyGrid applications communicating amongst themselves in order to resolve (future) scheduling conflicts.

References

[1] R. Buyya, editor. *High Performance Cluster Computing: Architectures and Systems*. Vol. 1. Prentice Hall, 1999.

[2] I. Foster and C. Kesselman, eds. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[3] The Globus Project. `http://www.globus.org`

[4] K. Kennedy *et al*. Toward a framework for preparing and executing adaptive grid programs. In *Proc. NSF Next Generation Systems Program Workshop (IPDPS)*, Apr 2002.

[5] The AppLes Project. `http://apples.ucsd.edu`

[6] The Cactus Code Server. `www.cactuscode.org`

[7] G. Allen *et al*. Gridlab: Enabling applications on the grid. In *Proc. of the 3rd International Workshop on Grid Computing*, LNCS 2536, pages 39–45, Nov 2002. Springer.

[8] R. Wolski, N. Spring, and J. Haye. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 1999.

[9] C. Boeres and V.E.F. Rebello. On solving the static task scheduling problem for real machines. In M. Fiallos *et al*, editors, *Models for Parallel and Distributed Computation: Theory, Algorithmic Techniques and Applications*, chapter 3, pages 53–84. Kluwer Academic, 2002.

[10] D. Culler *et al*. LogP: Towards a realistic model of parallel computation. In *Proc. 4th ACM Symposium on Principles and Practice of Parallel Programming*, May 1993.

[11] T. Kielmann *et al*. Network performance-aware collective communication for clustered wide area systems. *Parallel Computing*, 27(11):1431–1456, 2001.

[12] O.Sinnen and L. Sousa. A platform independent parallelising tool based on graph theoretic models. In *Proc. of the International Conference on Vector and Parallel Processing (VECPAR 2000)*, pages 154–167. Springer, 2000.

[13] The EasyGrid Project. `easygrid.ic.uff.br`