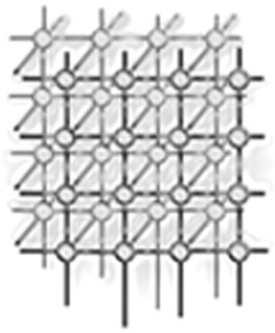
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2004; **16**:425–432 (DOI: 10.1002/cpe.821)

EasyGrid: towards a framework for the automatic Grid enabling of legacy MPI applications



Cristina Boeres and Vinod E. F. Rebello*,†

Instituto de Computação, Universidade Federal Fluminense (UFF), Brazil

SUMMARY

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 with ease remains a challenge. This paper outlines the EasyGrid methodology for the efficient and robust execution of (legacy) MPI programs across distributed computing clusters. The principal objective of this work is to identify the application-oriented middleware necessary for, as well as to develop a framework to automatically generate, *system-aware* applications capable of executing in dynamic, unstable, distributed environments such as computational Grids. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: Grid computing; system-aware applications; Grid middleware; MPI programs; task scheduling

INTRODUCTION

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 and according to demand to anyone that needs it [1]. Easy access to this computational power gives a much larger community of users than before the opportunity to solve computational problems. The realization of the vision, and thus the success of Grid computing, hangs on the community's ability to implement either an environment from which traditional applications draw performance with ease, or an environment to aid programmers in developing new applications capable of executing efficiently on the Grid.

*Correspondence to: Vinod E. F. Rebello, Instituto de Computação, Universidade Federal Fluminense, Bloco E, 3° Andar, Rua Passo da Pátria 156, São Domingos, Niterói, CEP 24210-240, RJ, Brazil.
†E-mail: vinod@ic.uff.br

Contract/grant sponsor: Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq); contract/grant number: 552205/2002-8

Received 10 August 2003
Revised 10 September 2003
Accepted 5 September 2003

One of the popular advantages of Grids is that they make computing power available to users who themselves have insufficient resources locally to execute their applications. Despite this, however, 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 behaviour encountered in Grid environments, developing Grid-enabled applications presents 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 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 currently being focused at *middleware* level [1]. The goal of middleware is to abstract over the complexity of Grid environments. Many of the middleware services being developed 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 not only require 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 developed or standardized middleware were available, it is questionable if typical programmers would be knowledgeable enough to write Grid applications capable of using the services appropriately. 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 fulfil 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 burden on the programmer or the user. Our work focuses on computational Grids (rather than service Grids or data Grids [1]) for the execution of applications which require large amounts of distributed or parallel computation. This work-in-progress paper presents a brief overview of a framework for the automatic Grid enabling of legacy MPI applications. We use the term *legacy* to refer to applications which were written without the Grid in mind. Named EasyGrid [2], the framework aims to address issues relating to an application's portability, execution efficiency and robustness to resource failures.

MOTIVATION AND RELATED WORK

The acceptance of innovative techniques for improving solutions to computational problems can be hindered by the fact that existing applications often need to be rewritten. Ideally, the same application should be capable of executing just as efficiently on conventional parallel machines (for example, a cluster of PCs or a supercomputer) as on a Grid without the need for modification.

New frameworks need to be created to make it easy for application developers to take advantage of new Grid capabilities. Realizing this objective requires two undertakings [3]: designing an interface which insulates programmers and users from the underlying intricacy of Grid environments without sacrificing the application's performance; and providing an execution environment that efficiently adapts the application to the dynamically changing resources of a Grid. Both of these undertakings are currently at the centre of Grid research on problem solving environments.

The current approach to application development focuses on implementing Grid-enabling computational frameworks (e.g. AppLeS [4], Cactus [5], the GrAD Project [3] and GridLab [6]). The principal objective of these frameworks is to provide abstractions to Grid services, allowing them to be accessed easily from within an application. Unfortunately, to execute on a Grid, existing applications need to be modified to interact with the appropriate service components within the framework. Thus, application developers need to be 'Grid-wise' or at least have a good understanding of the interfaces and the services available. Grid execution environments (e.g. Condor-G [7] and Legion [8]) offer applications *service middleware* to provide resource and data management facilities (monitoring, scheduling, fault tolerance, etc.) as well as *system middleware* functionality as provided by, for example, the Globus Toolkit [9] (security, file transfers, resource allocation, directory services).

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 [4]. 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 both *easier* and *efficient*. The EasyGrid Project aims to address these issues. The objective is to secure the realization of the Grid vision by making parallel computing on the Grid accessible to average 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. Similar in concept to SmartApps [10] (shared memory applications which contain a runtime library to dynamically and adaptively select compiler optimizations), the EasyGrid compiler embeds runtime functions (i.e. service middleware) into the application to monitor and optimize its execution. These *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. This idea can be extended to designing parallel applications which, depending on the characteristics of the resources available, dynamically alter the algorithm, implementation or optimization employed to solve the given problem.

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 difficult but necessary. The EasyGrid methodology aims to allow programmers to concentrate on how to exploit parallelism to resolve the problem and leave the *EasyGrid framework* to generate a system-aware version of the application capable of utilizing the Grid resources available to the user in the most appropriate manner. The framework will be used to validate a middleware approach directed at the application, to study the static and dynamic scheduling problem on computational Grids [11], and to integrate fault tolerance and scheduling strategies for system-aware applications.

When addressing Grids, most people adopt a *system-centric* viewpoint (see Figure 1(a)), i.e. the Grid consists of a set of resources upon which a pre-installed service middleware acts as a *Resource Management System* (RMS) to control the execution of Grid applications [12]. Responsible for maintaining an efficient utilization of the resources, the RMS monitors and analyses system-specific information, e.g. resource workload, network throughput etc. (metrics which are generally not application specific), to make decisions about which resources each application should use at

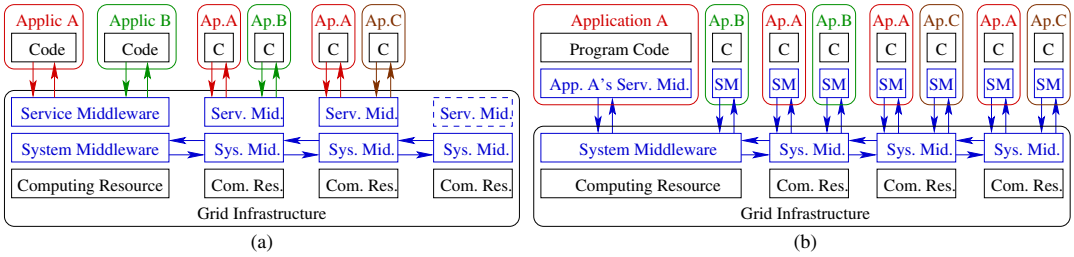


Figure 1. A simplified view of (a) system-centric Grid execution and (b) application-centric Grid execution.

each instant. However, relying on a system-oriented RMS to obtain good parallel program performance from a Grid, by solely adjusting processor workloads without considering the characteristics of the application, may not be sufficient or appropriate. A RMS is not suited to predicting the behaviour of each application which, given the increasing scale and use of Grid systems, is necessary for improving efficiency [13]. A better alternative might be to exploit the fact that each application can be made aware of its computational requirements and thus could adjust them according to the existing Grid environment in order to maximize performance, efficiency or resource utilization.

The EasyGrid methodology's view is *application-centric* (see Figure 1(b)), i.e. the service middleware is part of, and specific to, each individual Grid application. Efficient resource utilization is achieved by an *Application Management System* (AMS) distributed within each system-aware application. An AMS has the objective of matching its own application's requirements with what is available on the shared resources (Grid infrastructure). Each application takes the view that it has exclusive access to a (virtual) Grid. Since the resources are being shared, this virtual Grid actually operates at a dynamically changing fraction of its potential. 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. Application-centric execution allows an application to avoid being limited to resources with a specific service middleware, e.g. one resource in Figure 1(a) is unused since it lacks the appropriate pre-installed middleware. This in turn also facilitates the (dynamic) integration of resources into different Grid systems. Having access to a larger resource pool increases the chances for faster execution.

The EasyGrid framework

This framework has the objective of being able to *automatically* convert traditional parallel programs into system-aware programs 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. Good task and communication scheduling is crucial for the application to achieve acceptable performance. There is no consensus as to which scheduling model is most appropriate for the Grid. Thus our research is currently focused on the applicability of a *Heterogeneous LogP* (HLogP) model and the adoption of task replication-based scheduling algorithms [14] to minimize the effects of relatively high communication costs between

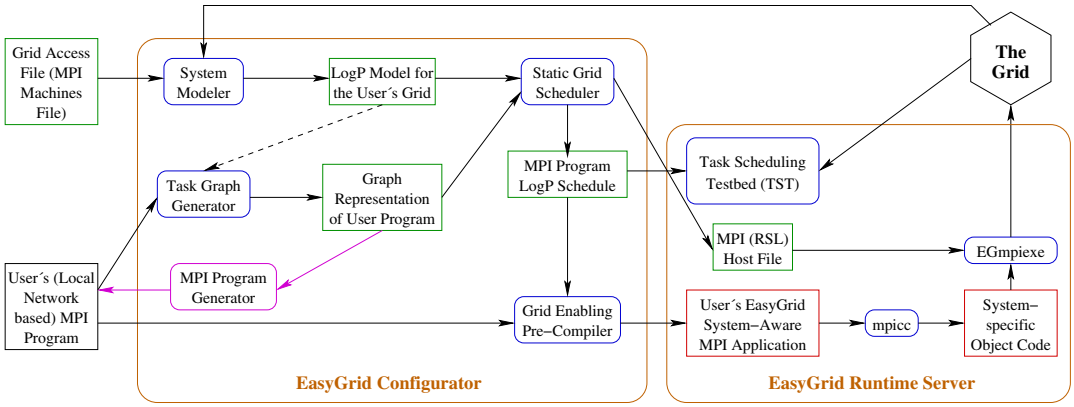


Figure 2. The EasyGrid framework.

Grid sites and to offer some degree of tolerance in the face of resource failures. (The *LogP model* [15] considers both communication overheads and limited communication capacity on networks, and has been shown to model a variety of systems including Grid-like wide area networks [16]).

Figure 2 presents an outline of the framework to generate an EasyGrid system-aware application from the user's MPI source code. 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 have access is also needed. The System Modeller creates a HLogP architecture model by obtaining information (e.g. processor speed, average and current load, communication latencies, etc.) either from a directory service (such as Globus's MDS [9]) or by direct measurement for each of the user's resources that are online. Predicting system behaviour on the Grid is an active research area and a number of Grid monitoring tools and techniques for estimating network and resource behaviour have been proposed [4,13]. While questions remain over the accuracy of the information obtained, this work aims to investigate how much this affects scheduling decisions.

Based on the characteristics of the available Grid resources and the application, an initial mapping or schedule is produced by the Static Grid Scheduler to guide the runtime execution. The MPI Host File, equivalent to a Globus RSL file [9] and generated by the scheduler, identifies the resources to which each MPI process 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 edges denote data dependencies (MPI communications). While the Task Graph Generator can be used to create a DAG representation of the user's program [17], 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 efficiency of the schedules generated and their execution in Grid environments. As well as an educational tool to compare and analyze schedules produced by different Static Grid Scheduler implementations, the Task Scheduling Testbed can be used as a portal interface to the EasyGrid

Runtime Server, which is capable of launching EasyGrid applications, collecting their AMS monitoring information and providing on-the-fly comparisons of their actual and statically predicted executions. The latter is useful to help determine the importance of the architectural model's accuracy and the benefits of the initial static schedule.

The Grid Enabling Pre-compiler generates an EasyGrid system-aware MPI application by using the scheduling information to restructure the user's original 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 a line of the user's source 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. Thus, the AMS functions are embedded as hidden MPI processes and integrated into the user's application by code wrapping the standard MPI function calls [18]. For portability reasons, no modifications are made to underlying MPI implementations. The script EGmpiexe takes care of security issues (e.g. setting up Globus's Grid proxies), makes sure each resource has the correct (operating) system-specific executable, and initiates the application's execution. The EasyGrid application with its AMS is distributed at three levels: a home or global manager node, responsible for the overall 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.

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 self-monitoring system [18] which provides information to other AMS functions (e.g. dynamic scheduling, fault tolerance). A side effect is the degree of intrusion suffered during the user program's execution. Currently, each local processing node monitors the MPI communications in the user code, passing this information to the site manager. As well as passing the information from all of its local processing nodes to the home node, the site manager's other AMS functions will use this information to make local decisions. An initial analysis based on synthetic MPI programs shows that EasyGrid AMS monitoring incurs less than 5% intrusion (around 1% on average) using MPICH-G2 [9] from the Globus Toolkit to execute on the GridRio Computational Grid [19].

SUMMARY AND ONGOING 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. However, deciding on an appropriate programming model for the Grid will take time. Since Grid resources are available today, this work aims to make running (legacy) parallel applications on the Grid easier, especially for non-Grid specialists (rather than making parallel programming easier). The goal of the EasyGrid methodology is to allow programmers to focus 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 and

testing one program for a locally available computing platform and another for the Grid. To achieve this we propose that Grid applications should be system-aware.

The EasyGrid methodology adopts an application-centric service middleware approach rather than the traditional system-centric one and aims to investigate the benefits in terms of application portability and efficiency. From a system-centric viewpoint, middleware is associated with the Grid infrastructure, whereas from an application-centric viewpoint the middleware is associated with, and tailor-made for, each application. In the case of the former, a RMS tries to decide which applications may or may not use a given resource at a particular moment. In the latter, each application decides which of the available resources to use, dynamically adjusting its execution to better exploit the current state of the Grid environment.

The effectiveness of integrated static and dynamic scheduling algorithms (based on communication models which consider both hardware and software overheads and optimization techniques like task replication [14]) is the focus of investigation [11]. Many functions associated with both system and service 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 and to evaluate how much this affects 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. A further issue is how other EasyGrid and native applications which share the computing resources influence another EasyGrid application's execution. There is scope to investigate the possibility of independent EasyGrid applications communicating amongst themselves in order to resolve (future) scheduling conflicts.

REFERENCES

1. Foster I, Kesselman C (eds). *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann: San Francisco, CA, 1999.
2. The EasyGrid Project. http://easygrid.ic.uff.br/ [10 August 2003].
3. Kennedy K *et al*. Toward a framework for preparing and executing adaptive Grid programs. *Proceedings NSF Next Generation Systems Program Workshop (IPDPS 2002)*, Fort Lauderdale, FL, April 2002. IEEE Computer Society Press: Los Alamitos, CA, 2002.
4. The AppLes Project. http://grail.sdsc.edu/ [10 August 2003].
5. The Cactus Code Server. http://www.cactuscode.org/ [10 August 2003].
6. Allen G *et al*. Gridlab: Enabling applications on the Grid. *Proceedings of the 3rd International Workshop on Grid Computing*, Baltimore, MD, November 2002 (*Lecture Notes in Computer Science*, vol. 2536). Springer: Berlin, 2002; 39–45.
7. The Condor Project. http://www.cs.wisc.edu/condor/ [10 August 2003].
8. Grimshaw A *et al.* The Legion vision of a worldwide virtual computer. *Communications of the ACM* 1997; **40**(1):39–45.
9. The Globus Project. http://www.globus.org/ [10 August 2003].
10. Dang F, Garzarán MJ, Prvulovic M, Zhang Y, Jula A, Yu H, Amato NM, Rauchwerger L, Torrellas J. SmartApps, an application centric approach to high performance computing: Compiler-assisted software and hardware support for reduction operations. *Proceedings 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, April 2002. IEEE Computer Society Press: Los Alamitos, CA, 2002.
11. Boeres C, Lima A, Rebello VEF. Hybrid task scheduling: Integrating static and dynamic heuristics. *Proceedings 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2003)*, São Paulo, Brazil, November 2003, Navaux P (ed). IEEE Computer Society Press: Los Alamitos, CA, 2003.
12. Krauter K, Buyya R, Maheswaran M. A taxonomy and survey of Grid resource management systems for distributed computing. *Software—Practice and Experience* 2002; **32**(2):135–164.
13. Wolski R, Spring N, Haye J. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems* 1999; **15**(5/6):757–768.

14. Boeres C, Rebello VEF. On solving the static task scheduling problem for real machines. *Models for Parallel and Distributed Computation: Theory, Algorithmic Techniques and Applications* (*Applied Optimization Series*), ch. 3. Fiallos M, Corrêa R, Dutra I, Gomes F (eds.). Kluwer: Dordrecht, 2002; 53–84.
15. Culler D, Karp R, Patterson D, Sahay A, Schauser KE, Santos E, Subramonian R, von Eicken T. LogP: Towards a realistic model of parallel computation. *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, May 1993. ACM Press: New York, 1993.
16. Kielmann T, Bal H, Gorlatch S, Verstoep K, Hofman R. Network performance-aware collective communication for clustered wide area systems. *Parallel Computing* 2001; **27**(11):1431–1456.
17. Sinnen O, Sousa L. A platform independent parallelising tool based on graph theoretic models. *Proceedings of the International Conference on Vector and Parallel Processing (VECPAR 2000)*. Springer: Berlin, 2000; 154–167.
18. Freire PMA. Monitoramento para aplicações MPI system-aware. *Master's Thesis*, Instituto de Computação, Universidade Federal Fluminense, Niterói, Brazil, 2003 (in Portuguese).
19. The GridRio Computational Grid. http://easygrid.ic.uff.br/grid/GridRio.html [10 August 2003].

