A Tool for the Design and Evaluation of Hybrid Scheduling Algorithms for Computational Grids

B.A. Vianna, A.A. Fonseca, N.T. Moura, L.T. Menezes, H.A. Mendes, J.A. Silva, C. Boeres and V.E.F. Rebello

Instituto de Computação, Universidade Federal Fluminense (UFF)
Rua Passo da Pátria, 156 Bloco E, Niterói, CEP 24140-240 RJ, Brazil

{bvianna, afonseca, nmoura, lmenezes, hmendes, jacques, boeres, vinod}@ic.uff.br

ABSTRACT

One of the objectives of computational grids is to offer applications the collective computational power of distributed but typically shared heterogeneous resources. Unfortunately, efficiently harnessing the performance potential of such systems (i.e. how and where applications should execute on the grid) is a challenging endeavor due principally to the distributed, shared and heterogeneous nature of the resources involved. This paper presents a tool to aid the design and evaluation of scheduling policies suitable for efficient execution of parallel applications on computational grids.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming – *Parallel programming*; I.6 [**Simulation and Modeling**] Simulation Support Systems – *Environments*

General Terms

Algorithms, Performance, Design, Experimentation.

Keywords

Task scheduling, Grid computing, Performance tool.

1. INTRODUCTION

The objective of computational grids is to coordinate shared, distributed heterogeneous resources to work as a single computational resource. The availability of such an environment opens new horizons for research in areas previously unexplored or limited for economic or impractical reasons.

The existence of long distance, low cost, high performance networks is encouraging the development of applications which take advantage of geographically dispersed resources. However, relatively few applications exploit the computational power available from such environments efficiently. Due to the diversity of resources, their dynamic behavior and the instability generally encountered in grids, developing applications capable of executing efficiently in such environments is still a challenge.

2nd Workshop on Middleware for Grid Computing, Toronto, Canada.

Users of parallel system frequently complain of the difficulty of achieving a reasonable fraction of the theoretical peak performance from the systems they use. In environments like grids with so much variation, it is extremely difficult for users (typically scientist and engineers with little system knowledge) to decide, for each application and for each execution, which resources would be the most appropriate. The challenge is to execute applications efficiently and robustly in grid environments, without placing this burden on the programmer or the user.

The EasyGrid methodology [3] aims to take up the challenge of relieving the programmer the task of enabling (existing) applications to execute efficiently in grid environments, and thus avoid the need to develop one version of the application for a local computing platform and another for the grid. This methodology is based on *application-centric* middleware which provides services tuned and oriented towards the individual application. The EasyGrid framework attempts to make the aspects related to the grid transparent to the programmer or user. To achieve this, parallel applications are transformed automatically into system-aware ones by incorporating application-specific middleware (AMS). These *system-aware applications* are adaptive, robust to resource failure (fault tolerant), self-scheduling programs capable of coping with the changes which occur and thus execute efficiently in shared, dynamic, unstable distributed environments like the grid.

Initially, the EasyGrid project is focusing on parallel applications which use the communication library MPI [14] due to its widespread use in scientific applications (although the middleware community is moving towards Java-based grid services, we believe that Java is still not the language of choice in the scientific community, and that there is also an urgent need to support existing (legacy) non-grid applications), and as an introductory tool to parallel programming.

Crucial to obtaining an acceptable performance is a good allocation of application processes to the processors available. The problem of developing appropriate scheduling heuristics for the execution of parallel applications on computational grids involves considering two distinct but related phases: a static scheduler should be employed to initially allocate processes or tasks to a selected group of the resources available; and due to the likely fluctuating behavior of grid resources, a dynamic scheduler should adjust the task allocation during the application's

execution in order to exploit the grid environment better. One approach to reduce the scheduling overhead of the dynamic scheduler is to make use of information provided by a static scheduler (*hybrid scheduling*) [2,12].

While numerous scheduling algorithms exist for the static scheduling problem, and for the dynamic scheduling problem respectively, it is difficult to find a strategy which always gives the best result (e.g. smallest execution time) irrespective of the application's characteristics (e.g. if interprocess communication occurs, computation to communication ratio, topology).

This paper presents a graphical tool (called the ***Task Scheduling Testbed***) which aims to ease the development and analysis of task scheduling algorithms for system-aware applications. The principal objectives of the tool can be summarized as follows:

- Facilitate the investigation of scheduling approaches and (combinations of) techniques (for example task priorities, processor choice, and task replication) in the **design** of heuristics appropriate for grid environments.
- **Evaluate** candidate algorithms in terms of the chosen criteria, for example, the *makespan* (execution time of the parallel algorithm) and the number of processors necessary. Experimental (as opposed to theoretical) evaluations require comparisons to be made with existing heuristics for a range of environmental conditions.
- The schedules generated by the proposed heuristics should be **validated** either by independent simulation or by execution of the application in a real grid environment.

2. HYBRID SCHEDULING

Many parallel applications can be represented by directed acyclic graphs (DAGs), where the vertices model the tasks and the edges, data dependences between tasks pairs, and consequently communication. In general, when two adjacent tasks are allocated to the same processor, the cost associated to the respective edge is considered negligible. Therefore, in order to minimize the application's completion time, a tradeoff between exploiting parallelism and minimizing communication costs must be found. The communication characteristics considered relevant are often defined in a *communication model*.

The problem of scheduling the tasks of an application so that the execution time is minimized is NP-complete. This has led to development of numerous heuristics which attempt to find good quality schedules quickly. The majority of these proposals are based on the standard communication model used by the scheduling community, the *delay model*, where the sole communication parameter is the transit time for each message transmission [15]. Research has shown, however, that this model is not appropriate for today's computing platforms and thus has motivated the development of new performance models such as *LogP* [5]. The *LogP* model considers characteristics such as: the number of processors; the processing cost to send or receive a message; the transmission delay of a message between processors; and the maximum rate that messages can be sent or received. These parameters help represent the dominant communication characteristics of each individual communication as well as the fact that networks have a limited communication capacity. As there still no consensus as to the model most appropriate for grids, this works adopts the use of a new scheduling model called Heterogeneous *LogP* (*HLogP*) [13], suitable for MPI applications in heterogeneous computing environments.

Recently, the static scheduling community has been focusing on the problem of scheduling tasks onto heterogeneous resources due to the advent of cluster and grid computing. The majority of heterogeneous scheduling heuristics [16] belong to the class of *list scheduling* algorithms [7,11] probably due to the combination of low complexity and reasonable quality schedules (in terms of the *makespan* and the use of limited numbers of processors).

However, in shared environments, predicting the processing power and communication bandwidth available to a given application is difficult which makes designing efficient and effective static scheduling algorithms extremely challenging. The estimations assumed by the static scheduler may no longer be the same during run-time and so cause the application to perform poorly [12]. To overcome this handicap, *dynamic schedulers* use local information available at run-time to make their scheduling decisions. These decisions, however, need to be made quickly in order to avoid stalling program execution while waiting for tasks to be scheduled, thus minimizing the impact (i.e. the scheduling overhead) on the execution time of the application.

In order to reduce the dynamic scheduling overhead and to improve the application performance even further, a hybrid scheduling approach should be applied [2,12]: global information produced by a static scheduler (based on estimations of characteristics of the application and target architecture) is used in the decision making of the dynamic scheduler. The static phase of the hybrid scheduler is used to tune the application to an *initial* configuration of the target system, while the dynamic phase, adapts the allocation to the variations which occur during execution.

2.1 Designing Hybrid Scheduling Algorithms

2.1.1 Configurable List Scheduling

In a *list scheduling* heuristic, the application tasks are ordered in a list in accordance with a given priority. This priority may be calculated once only before, or recalculated during (so called *dynamic priority*), the scheduling process. Then, a series of iterations are performed so that each task is scheduled in order onto a given processor. At each iteration, two selections are made: *task choice*, where the task with the highest priority is chosen; and *processor choice,* to identify the processor to which the selected task will finish the earliest. The main difference between the various list scheduling algorithms available in the literature is the criteria used in each of the selection processes [11].

The *configurable list scheduling* algorithm developed in this work is designed for heterogeneous systems, modeled by *HLogP,* with a limited number of processors. A number of mechanisms are offered as options for the *testbed* user to configure a distinct list scheduling scheme. The current options available are:

- *Task choice priority* — a variety of priorities (*top level, bottom level, ALAP, CP,* and others [9]) already used in homogeneous scheduling were adapted to be used to heterogeneous scheduling, using the average computing cost of a task considering all of the processors [16];
- *Dynamic priorities* — when tasks are allocated to the same processors, the communication costs between them are zeroed. Therefore, updating the priorities to consider which costs have been eliminated can provide a better indication of the relative importance of unscheduled tasks in future iterations of the scheduling process;

- *Multiple priorities* — when more than one task has the same priority, a second (and third) priority can be defined to narrow down the selection. The main objective of defining multiple priorities is to fine tune the task selection process since this is crucial to achieve a good quality solution;
- *Restricting the number of processors* — ties can also happen when selecting a processor. Selecting a *new* processor may lead to a better solution in future iterations. However, this may also increase the number of processors necessary to implement the resulting schedule;
- *Task insertion in idle processor periods* — during the scheduling process, it is possible for idle periods of time to exist between tasks already scheduled. Scheduling new tasks during these periods (obviously, respecting the precedence relationship and also, without delaying already scheduled tasks) can lead to better solutions particularly when the degree of processor heterogeneity is high [16];
- *Pos-scheduling resource elimination* — attempts to reduce further the number of processors necessary by moving scheduled tasks to processors capable of executing them without increasing the *makespan*.

2.1.2 Dynamic Scheduling Strategies

Dynamic schedulers can be distinguished by the mechanisms employed to trigger *scheduling events* during the application's execution and the priorities used to schedule tasks during each event. The dynamic schedulers available in the tool use information generated by the chosen static scheduler to *reallocate a subset* of application tasks onto the available processors. Reallocation takes into consideration the current characteristics of the system. At each scheduling event, like a list scheduler, two phases are executed: the *task choice* phase; and the *processor choice* phase. Note that the priorities adopted by the dynamic scheduler are independent those of the static scheduler. These two phases are repeated until all of the tasks in the subset have been (re-)assigned processors. Currently, four pairs of *task* and *processor choices* are available [2]:

- *Minimal Partial Completion Time Static Priority* (**PS**): The task choice selects the task with the highest *scheduled bottom level* (obtained from the static scheduler) while the processor choice re-allocates this task to the processor that minimizes its finish time.
- *Minimal Completion Time Static Priority* (**CS**): The task choice selects the next task with the highest scheduled bottom level while the processor choice re-allocates this task to the processor that minimizes the task's critical path.
- *Minimal Completion Time Dynamic Priority* (**CD**): The task choice selects the next task with the highest associated critical path cost while the processor choice re-allocates this task to the processor that minimizes the task's critical path.
- *Minimal Completion Time Bottom Level Graph Priority* (**CB**): The task choice selects the next task with the highest bottom level while the processor choice re-allocates this task to the processor that minimizes the task's critical path.

Currently, in the hybrid scheduling framework, the frequency at which scheduling events occur is related to the partitioning of the input DAG in *blocks of tasks*. The following definition variations can be applied to each of the dynamic priority pairs:

- Each block contains all the tasks of a given level [12]. The scheduling event for block *k* occurs when the first task of block *k-1* has been executed;
- Each block contains tasks from *N* distinct adjacent levels, and again the scheduling event for block *k* occurs when the first task of block *k-1* has completed its execution;
- Block 0 has the tasks from level 0 and the remaining blocks each contain tasks from *N* distinct adjacent levels. The scheduling event for block *k* occurs when the first task of the last level of block *k-1* has finished its execution.

2.2 Evaluating the Scheduling Algorithms

Due to complexity of the scheduling problem, especially when considering heterogeneous resources, determining the optimality of an algorithm is often hard. This has lead researchers to attempt to analyze specific classes of DAGs, be it in accordance with the topology (e.g. join, forks, trees, diamond, or irregular) and/or with the *granularity*. Typically, the graph topologies chosen represent specific classes of applications, while varying the granularity can also account for a variety of target systems. For a complete evaluation of a heuristic, experimental comparison with existing state of the art algorithms, for a wide range graphs and architectural instances, is fundamental. The following well known scheduling algorithms have been implemented (in some cases modified) and included in the *testbed* (others can be added easily). Also included are two suites of benchmark scheduling graphs [1] and [10] which contain both regular (various sizes of trees and diamond graphs) and irregular graphs (randomly generated and graphs taken from the literature) as well graphs of applications.

The Dynamic Critical Path (DCP) algorithm [9] was designed for the problem of scheduling tasks on an unbounded number of homogeneous processors and has the best overall performance of all list scheduling algorithms [10]. In the testbed, DCP was adapted to schedule DAGs onto a limited number of heterogeneous resources. The *Heterogeneous Earliest Finish Time* (HEFT) is considered one of the best algorithms for scheduling tasks onto heterogeneous processors [16]. The *Earliest Time First* (ETF) algorithm is a very well-known list scheduling algorithm [7] originally proposed to schedule DAGs on a limited number of homogeneous processors. We adapted ETF to consider a limited number of heterogeneous resources. The *Level Duplication Based Scheduling* algorithm (LDBS) [6] is a list scheduling approach which uses replication to schedule a DAG onto a heterogeneous system.

In the literature, few algorithms have been designed to tackle the scheduling problem under the *LogP* model. The majority of these are clustering algorithms [1]. The only *list scheduling* approach designed to deal with the *LogP* parameters was proposed by [8] as a modified ETF. Using the same technique, we adapted HEFT and with new techniques adapted the configurable list scheduling algorithm to the *HLogP* model.

In order to be able to analyze the performance of the hybrid scheduling strategies on various grid architectures, the testbed uses the SimGrid simulation tool [4]. SimGrid simulates the execution of distributed applications in grids, providing a means to evaluate both static and dynamic scheduling algorithms. SimGrid defines entities that represent the tasks and their dependencies in a given application as well as the resources (processors and communication links). In order to implement the dynamic scheduling approaches, the testbed provides SimGrid with the schedule of a *block* of application tasks to execute, and indicates the condition which triggers the next scheduling event.

3. THE TASK SCHEDULING TESTBED

Although the *Task Scheduling Testbed* is primarily a tool for designing and evaluating static and hybrid scheduling strategies for distributed heterogeneous systems, it is also capable of acting as a grid portal to automatically (i.e. without user intervention) schedule MPI applications onto Globus Toolkit (2.4) based computational grids. The tool can be used in four distinct phases:

- defining the application(s) to be scheduled and the characteristics of the target architecture, and designing or choosing a scheduling algorithm;
- analyzing the chosen static heuristic by comparing the schedules generated with others available in the tool over a set of DAGs and architectural models chosen by the user;
- evaluating the hybrid scheduling heuristic using SimGrid;
- executing (and monitoring) a scheduled (system-aware) application on a computational grid.

3.1 Designing a scheduling algorithm

The goal of the tool is to help define an appropriate scheduling strategy (a static schedule for the initial placement on the grid and hybrid algorithm to be incorporated into the AMS) specific for each user application to transform it to a system-aware one.

Initially, the user is asked whether (s)he would like to schedule a real MPI user application or a synthetic one represented by a DAG from either of the two benchmark suites [1,10] or one defined by the user. Using the *graphviz* package from AT&T Research Labs, the user can visualize the DAG's structure with the weights associated to the tasks and edges. An example is shown in Figure 1. Future work aims to investigate how to generate DAGs for real MPI applications.

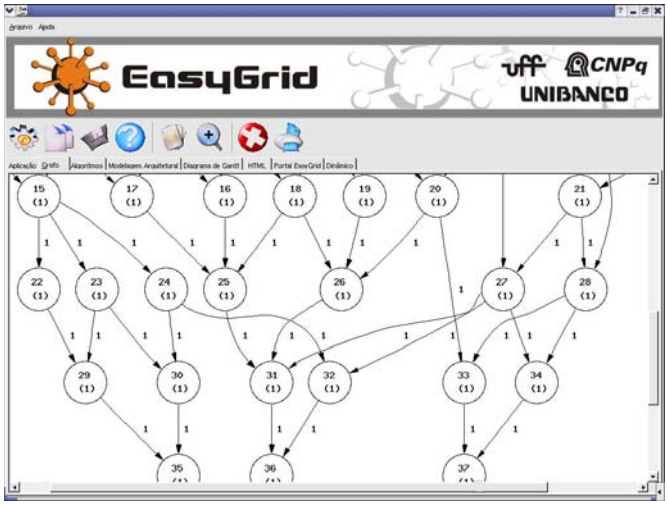

Figure 1: DAG representing an application.

The architecture model can also be user defined (i.e. synthetic values read from a file) or be of a real cluster or grid. Modeling a grid is carried out in a number of steps (Figure 2):

- The tool uses a *Grid Universe File* (maintained by a grid administrator) which contains information on all grid resources such host names, which MDS node they report to, which use a NFS server, which MPI is installed (MPICH-G2, MPI-LAM). This is matched to a list of resources the users has access to, and the MPI version of the application, in order to present the user with list of candidate resources.
- The tool then ascertains the status of the candidate resources by accessing the respective MDS servers to retrieve technical details, and verifies if a *globus gatekeeper* process is running and accepting jobs. If not, the node is identified as inactive.
- The user is then given the option to modify the list before the tool attempts to calibrate the *HLogP* model for the chosen virtual grid. A valid grid-proxy is required to execute the grid modeler (an MPI program that measures the computing power available to the user on each node and communication costs between nodes [13]).

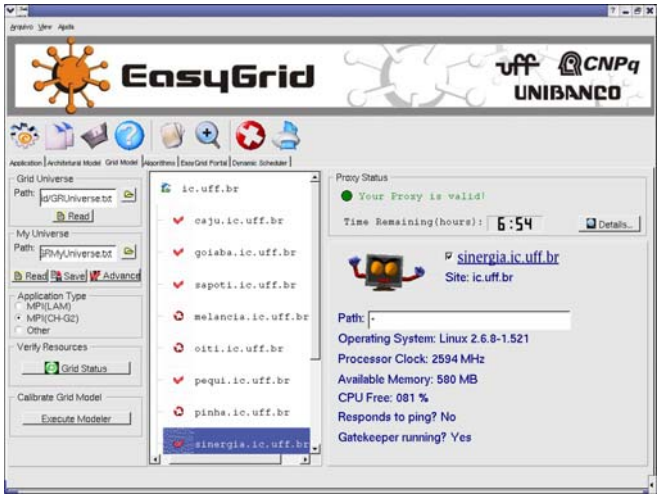

Figure 2: Verifying the status of grid resources.

A friendly interface offers the opportunity for the user to choose various options when developing and analyzing different scheduling strategies to efficiently schedule the chosen application on the modeled system.

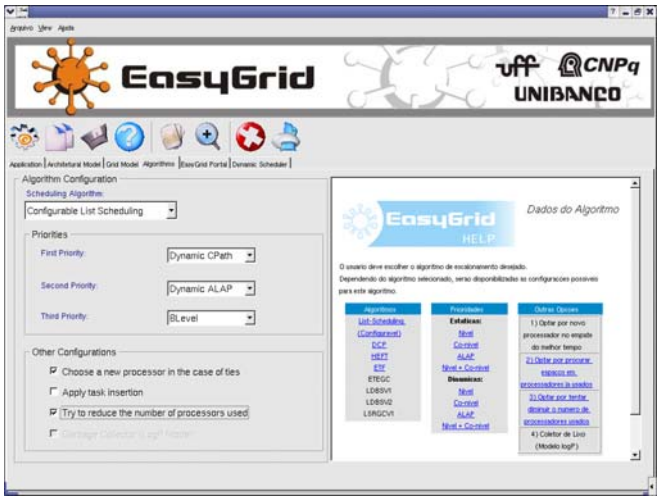

Figure 3: Designing a static scheduling heuristic.

As discussed in Section 2.1.1, for static schedulers, the user can define one or more versions of *Configurable List Scheduling* as well as use heuristics taken from the literature (see Figure 3). In the case of hybrid schedulers (Section 2.1.2), the user not only selects the static scheduler but also, the dynamic strategy (PS, CS, CD and CB schemes) and the scheduling event variant (Figure 4).

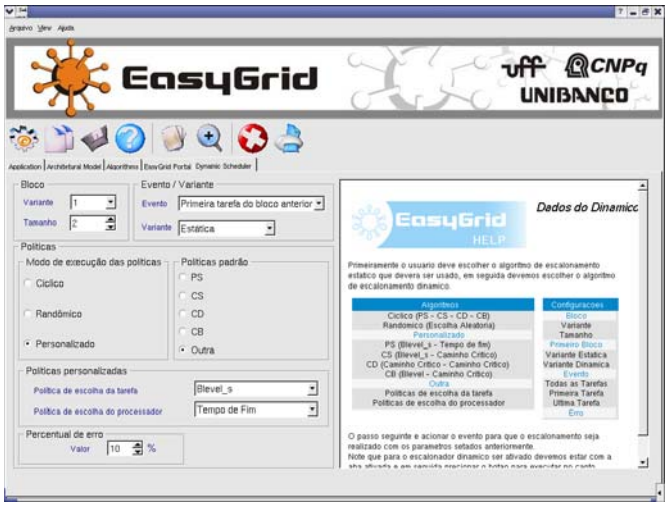

Figure 4: Defining the hybrid scheduling heuristic.

3.2 Evaluating scheduling algorithms

Given the specification of the scheduling strategy and its input data, the results can be evaluated graphically. The schedule produced can be visualized in the form of a Gantt chart. In Figure 5, each line corresponds to a processor and each block, to a task or overhead with a width proportional to its execution cost.

It is possible to compare various scheduling algorithms simultaneous using a group of selected DAGs. A table *Algorithm* versus *DAG* shows, for each graph, the makespan obtained and the number of processors required by the schedule produced by the respective heuristics. Also available is a table showing a pair-wise comparison between the strategies under evaluation in terms of the number of worse, equal and better schedules, and the percentage of times that each algorithm produces a schedule with the smallest makespan. The user can save the resulting schedules, evaluations and Gantt charts.

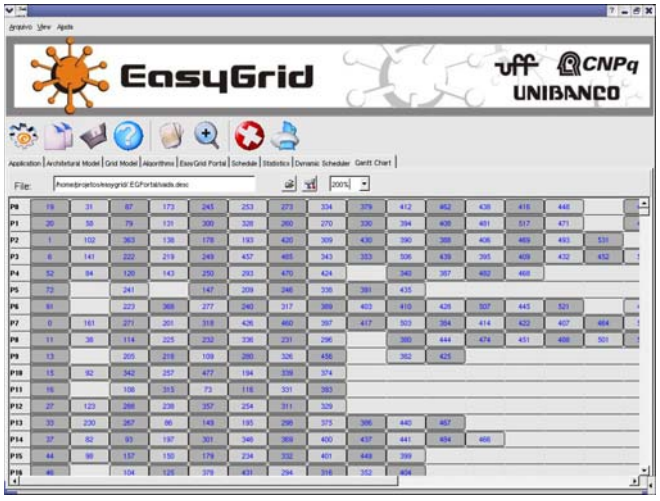

Figure 5: Gantt chart of a resulting schedule.

3.3 Validating the schedules

In order to validate both static and dynamic schedules, it is necessary to execute the application in a real environment or to simulate the execution (currently simulation is the only option available for evaluating the hybrid schedulers). The advantage of a simulated environment over a real environment is the ability to evaluate different architectural characteristics, although the value of the evaluation depends on the accuracy of the model adopted by the simulator. Precise results can be obtained for the execution on a real grid but the conclusions only apply to that grid at the time of the execution.

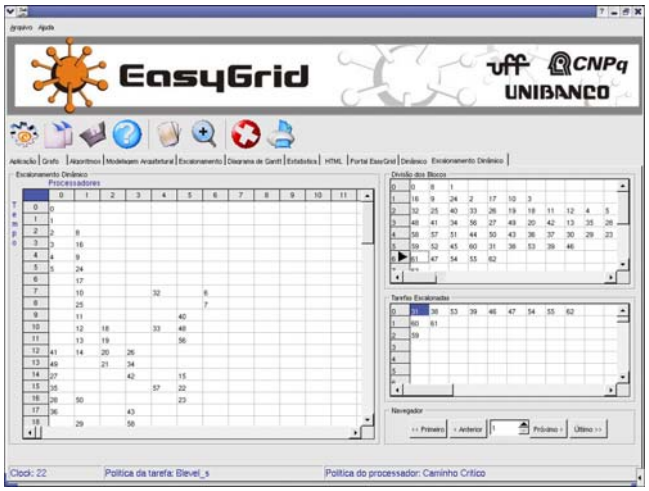

Figure 6: Simulating a hybrid schedule with SimGrid.

Figure 6 presents the simulated execution of an application. The user can step through the simulation which shows which task have been executed (left hand side), which tasks are waiting in the respective processor queues (bottom right) and which *block of tasks* is currently being scheduled (top right).

Figure 7: The EasyGrid Portal.

Executing MPI applications in computational grids requires a sequence of steps which probably appears complex and monotonous to inexperienced or casual users. Therefore, to ease this effort, the functionality of a grid portal has been integrated into the scheduling tool (as shown in Figure 7). The first pass to executing an application is the creation of a grid-proxy (requires a user certificate and password) which permits access to the grid resources. The proxy has a period of validity which can be managed by the portal.

If the user chose to a schedule a DAG, an equivalent synthetic MPI program will be generated. The user can choose to include in the source code of the application (real or synthetic) the AMS middleware (currently only the self-monitoring service is available, the dynamic scheduling and fault tolerance services are being developed) and provide any necessary compilation options. The user is also queried with regard to transferring code to remote grid resources. The tool uses the *Grid Universe* file to determine which resources require a local copy of the executable, and also which require the source code to be compiled locally.

If the user modeled the grid earlier, (s)he has the option to choose between a schedule determined by the algorithms within the tool or a traditional MPI round-robin schedule. Otherwise only the latter option is available. The appropriate RSL file is then generated to launch the application. If the AMS middleware has been inserted, the execution of the application can be monitored in the form of a Gantt chart similar to Figure 5.

4. CONCLUSIONS

The development of system software, tools and parallel applications for grid environments is challenging due to the dynamic and heterogeneous nature of the grids. The aim of a scheduling algorithm is the efficient execution of a parallel application considering the relevant characteristics of the target system. The *Task Scheduling TestBed* is a user friendly environment for the development and analysis scheduling algorithms for grids. The tool also integrates portal functionalities to launch and monitor the execution of system-aware MPI applications on Globus-based computational grids.

The modular design of the tool permits extensibility. Each scheduling algorithm and grid portal function is implemented as a separate executable or script, all coordinated by the graphical interface. The only drawback of the existing implementation is the necessity for the function modules, the interface and the user's applications to be installed on the same grid resource. Through a Java-based implementation of the interface which executes locally and communicates via web-services to a grid node capable of executing the function modules, a new version of the tool will allow mobile users to remotely access and launch applications from any computing resource with internet access.

This work has focused on scheduling *parallel* applications rather than distributed (e.g. bag of task or parameter sweep) ones since we consider the strategy more general and thus applicable to distributed applications as well as grid workflow systems.

5. ACKNOWLEDGMENTS

This work is funded by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under grant no. 552205/2002-8 and by Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ) under grant no. E-26/170.696/2004. B. Vianna is funded by an undergraduate Iniciação Científica studentship from UNIBANCO, A. Fonseca and L Menezes are funded by Iniciação Científica studentships from CNPq, N. Moura is funded by a Iniciação Científica studentship from CNPq/PIBIC (UFF). J. Silva is funded by a postgraduate Iniciação Tecnológica Industrial studentship from CNPq. C Boeres and V. Rebello are partially funded by research grants from CNPq.

6. REFERENCES

[1] C. Boeres and V.E.F. Rebello. Towards optimal task scheduling for realistic machine models: Theory and Practice. *The International Journal of High Performance Computing Applications*, 17 (2):173-189, 2003. Sage Publications.

[2] C. Boeres, A. Lima and V.E.F. Rebello. Hybrid Task Scheduling: Integrating Static and Dynamic Heuristics. In *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2003)*, São Paulo, Brazil, November 2003. IEEE Computer Society Press.

[3] C. Boeres and V.E.F. Rebello. EasyGrid: Towards a framework for the automatic grid enabling of legacy MPI applications. *Concurrency and Computation: Practice and Experience*, 16 (5): 425-432, 2004. John Wiley and Sons.

[4] H. Casanova, SimGrid: A Toolkit for the Simulation of Application Scheduling. In *Proc. 1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2001.

[5] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, USA, May 1993.

[6] A. Dogan and F Ozguner. LDBS: A duplication based scheduling algorithm for heterogeneous computing systems. In *Proc. of International Conference on Parallel Processing (ICPP'02)*, pages 352-358, Vancouver, Canada, Aug 2002.

[7] J. Hwang, Y. Chow, F. Anger and B. Lee, Scheduling precedence graphs in systems with interprocessor communications times. *SIAM J. Computing*, 18(2):1-8, 1989.

[8] T. Kalinowski,, I. Kort, and D. Trystram, List scheduling of general task graphs under LogP. *Parallel Computing*, 26(9):1109-1128, 2000.

[9] Y-K Kwok and I. Ahmad. Dynamic Critical–Path scheduling: an effective technique for allocating tasks graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506-521, May 1996.

[10] Y. K. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381-422, December 1999.

[11] Y-K Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4), Dec. 1999.

[12] M. Maheswaran and H. J. Siegel. A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems". In *Proc. 7th Heterogeneous Computing Workshop (HCW98)*, pages 57-69, Orlando, Florida, March 1998. IEEE Computer Society Press.

[13] H.A. Mendes, *HLogP*: *Um modelo de escalonamento para a execução de aplicações MPI em grades computacionais*, Master's Thesis, Instituto de Computação, Universidade Federal Fluminense, 2004. (In Portuguese)

[14] Message Passing Forum. *MPI: a Message Passing Interface*. Technical report, University of Tennessee, 1995.

[15] C.H. Papadimitriou, and M. Yannakakis, Towards and architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, v. 19, p. 322-328, 1990.

[16] H. Topcuoglu, S. Hariri, and M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems 13,(3): 260-274, March 2002.