A Flexible Architecture for Scheduling Parallel Applications on Opportunistic Computer Networks[⋆]

Marco A. S. Netto[1‡], Alfredo Goldman[1], Pierre-François Dutot[2]

{netto, gold}@ime.usp.br, dutot@loria.fr

[1]*Depart. of Computer Science, University of São Paulo, Brazil*

[2]*UHP, Nancy 1, LORIA[∗], France*

Abstract

Opportunistic computer networks are a promising environment to execute parallel applications due to the processing power supplied by the current desktop machines. Parameter sweep and master-worker applications are suitable for these environments since a failure of a process does not necessarily suspend the overall application. On the other hand, in tightly-coupled parallel applications a failure in a machine may interrupt or crash the entire application. Thus, different application classes require different scheduling strategies. Furthermore, in the context of Computational Grids, each Grid domain has its own administrative policies, user and application profiles and resource properties. All these reasons have motivated us to propose a flexible and extensible architecture for scheduling parallel applications which relies on different services designed as plug-ins. We also discuss the integration of this scheduling architecture in the InteGrade project which is a novel middleware infrastructure for Grid Computing.

1 Introduction

Despite the enormous evolution on the computers, several research areas such as Bioinformatics, Physics, Astronomy, Earth and Life Sciences make use of software systems that are always in need of an increasing amount of computing power to address complex problems. Grid Computing [9] is a technology that has been well investigated by researchers interested in executing complex applications that demand computer resources, such as desktop machines, clusters, supercomputers, scientific instruments and software systems, that are spread over several administrative domains.

Computer clusters are a well-known alternative to supercomputers to achieve the high processing power required with reasonable cost. However, although clusters provide a very

[⋆] Research supported by a grant from CNPq, Brazil - grant number: 55.0094/2005-9

[∗] LORIA (UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP)

[‡] Supported by a CAPES fellowship.

good cost performance ratio, the computer resources needed to execute complex applications may be unavailable in a single domain. Another approach to achieve high processing power required by complex applications is by harnessing idle CPU cycles of desktop machines. This is a low cost solution since both academic and enterprise environments hold several powerful desktop machines that are underutilized most of the time.

An efficient scheduling scheme is one of the most important issues to make a good use of computer resources. Scheduling parallel applications on dedicated resources is a complex problem, and such a complexity increases significantly when we consider that these resources are non-dedicated.

In order to execute applications on opportunistic computer networks, the scheduler should take into account that the machines are non-dedicated and hence interruptions to the ongoing applications may be frequent. Depending on the application type, these interruptions will result in different effects on the current schedule and on the execution process.

A well-known type of application is the Parameter Sweep Application (PSA), which involves the execution of the same program over a range of different parameter values [3]. PSAs are generally used in Computational Grids mainly due to the large number scientific applications that require the execution the same experiment on different datasets. Furthermore, as the experiments are independent, if a machine that is executing one experiment becomes unavailable, the execution of the other experiments may continue. As soon as one experiment finishes or a machine becomes available, the interrupted experiment can restart its execution - from scratch or from the last previous stored stage.

As opposed to PSAs, tightly-coupled parallel applications involve the execution of a set of processes that need to exchange information both asynchronously and synchronously. These applications are generally implemented using communication libraries such as Message Passing Interface (MPI) [11], Parallel Virtual Machine (PVM) [18] or Bulk Synchronous Parallel (BSP) [20]. This type of application is rarely used in opportunistic computer environments since a failure in a machine may interrupt or crash the entire application, and therefore the amount of wasted work is much more significant in relation to a loss in PSAs.

Thus, as scheduling parameter sweep and tightly-coupled parallel applications require different strategies, it is important to use a flexible and extensible scheduler so as to deal with different class of applications. In this work we present an architecture for scheduling parallel applications on opportunistic computer networks that relies on services that are used as plug-ins by the scheduler. Here we cover aspects of both online and batch scheduling strategies, use of preemption to allow malleability of parallel applications, as well as the use of resources availability prediction for a more efficient mapping of user applications. Furthermore, we will discuss how different scheduling strategies could be applied to execute different types of applications, in particular, parameter sweep and tightly-coupled parallel applications. At this stage of the work, our main focus is on the dynamicity problem. Nevertheless, in this paper we provide an overview of the possible scheduling strategies to deal with both heterogeneity and scalability problems. Moreover, as case study, we describe how to employ the proposed scheduler into InteGrade, a Grid middleware infrastructure that has been developed by our research group aimed at leveraging idle CPU cycles of desktop machines [10].

The remainder of this paper is organized as follows. In Section 2 we motivate our work based on some related work regarding the execution of parallel applications on dynamic environments. Following, in Section 3 we present in detail the scheduler architecture, and we also provide an overview of important services to assist the scheduler. In Section 4 we discuss some scheduling strategies based on these services. In Section 5 we illustrate the use of the proposed architecture in the InteGrade Grid middleware. Finally, in Section 6 we close the paper with a conclusion and proposals for future work.

2 Related Work

Exploitation of non-dedicated machines is a well studied subject due to the vast amount of processing power that the current desktop machines can supply. As consequence there are several projects with different goals [1–4, 13, 16, 19, 22]. Condor is one of the most well-known software system to harness idle CPU cycles of desktop machines [19]. This batch system allows migration of applications, with some limitations, among machines through the use of a Condor's library. It schedules user applications based on matching between computing resources and application requirements. Condor focuses mainly on execution of independent sequential applications. However, it provides services to allow the execution of multiple tasks with dependencies in a declarative form, as well as an interface with PVM called Condor Application Resource Management Interface (CARMI) [17]. CARMI permits PVM applications to dynamically allocate and release resources depending on the availability. These PVM applications are developed through the master-worker model [12]. Condor's researchers also propose scheduling strategies to deal with the impact on execution of these master-worker applications through the use of additional machines and task replication [13].

Another project related to the scheduling of applications is the Application Level Scheduler (AppLeS) [2]. AppLeS users develop their applications on a framework that provides services for scheduling tasks and load balancing at the application level. The scheduling algorithms in AppLeS rely on short-term prediction provided by the Network Weather Service [21]. Although such an approach can provide good performance results, the modification of existing applications consumes time and human resources that may not be available. Moreover, users may not have access to application source codes.

In [22] the authors describe adaptive scheduling strategies based on both application-level and system-level performance prediction aimed at heterogeneous non-dedicated machines. Their work focuses on parameter-sweep applications. Casanova *et al.* [3] presents heuristics for scheduling PSAs in Grid environments taking into account file sharing issues.

Currently most research projects that investigate the problem of scheduling applications on opportunistic computer environments work with parameter sweep and master-worker applications. Moreover, some of them require modifications on the user applications. In this context, our work is different since we also consider the scheduling of different kinds of applications on dynamic environments that do not require modification on the application source codes.

Furthermore, in relation to the number of processors an application utilizes, our work focuses on three classes: Rigid, Moldable and Malleable [8]. Rigid applications require a fixed number of processors to perform the work. Moldable applications are more flexible; the number of processors can be configured, but only at the time the application starts. Finally, malleable applications can be executed on different number of processors along the time. The scheduler can then shrink or expand the application during its execution according to the resource availability provided by the environment. Consequently, the scheduler has more flexibility to decide where to assign the applications, and it can change the assignment dynamically.

3 The Scheduler Architecture

When we are dealing with the scheduling of parallel applications on distributed non-dedicated resources, we need several services to assist the scheduler to make good decisions. For this reason, the proposed architecture for the scheduler relies on several services that can be dynamically plugged in to provide more information in order to improve user application scheduling decisions (Figure 1). To exemplify, besides the scheduling algorithms, some important services are resource prediction, migration cost definitions, checkpoint management, resource and application monitoring and inter scheduler communication. Following we present an overview of some of these services and how they could be exploited.

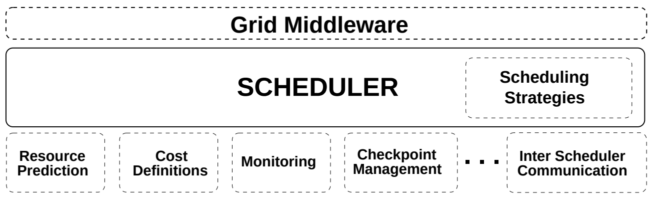

Figure 1: The scheduler module relies on plug-ins that provide services such as resource prediction, migration cost definitions, checkpoint management, inter scheduler communication, among others.

3.1 Support services

Resource prediction [15] is a fundamental tool for scheduling applications on non-dedicated machines. It can avoid the assignment of applications to resources that will be unavailable in a short period of time, thus reducing wasted work. Based on historical data, it is possible to generate resource usage patterns to allow the scheduler to best decide where and when applications should be executed.

When a user submits an application to the scheduler or when an application rescheduling is needed, the scheduler can rely on resource prediction service to discover the period that the current available machines could be used, and a list of machines and periods in such a way that they will be available in the future.

Fault tolerance is another essential service since machines may leave out the network at any time, and thus failures are a rule and not an exception. In this case, at specified time intervals, each machine must send keep-alive messages to the server. The server then removes a machine from its internal structures if one of them does not send a message in a time interval. Note that resources are not the only point of failure in systems. The user applications can fail and hence a fault tolerance service could report failures to the scheduler.

Monitoring application executions is also an important service to be considered. For example, if an application is executing for a long time and it has not finished yet, or no checkpoint was generated, the scheduler can make a decision to provide more resources to that application. Furthermore, the monitoring service can be used to supply important information for future application performance analysis.

3.2 Checkpoint Management

Checkpointing is a very useful feature available in some applications or provided by software systems and libraries. Briefly, it enables an application to be restarted from its last stored execution state [7]. In the context of opportunistic computer networks, together with the decrease of wasted work when an application needs to be interrupted, checkpointing can be exploited to preempt applications to other resources, as well as to enable the execution of malleable applications.

There can be two possible interactions between the scheduler and the checkpoint chosen strategy: (i) scheduler puts into action the checkpoint; and (ii) checkpoint puts into action the scheduler. The first case relies on the capability of the system or application to generate checkpoint at any time. However, such a possibility is not usual in applications - and it is usually done by system level - and it is a complex procedure. Furthermore, it results in several limitations - for example, it may be only possible to restart an application on the same machine architecture where it was checkpointed. In the second approach, we assume that applications frequently generate checkpoints, which is an usual feature available in complex computer simulation tools. A checkpoint manager can be implemented to, for example, report the scheduler that an application has recently generated, and based on this information the scheduler could make a decision - as we will describe in detail in Section 4.

The knowledge about the checkpoint frequency could assist the scheduler in the same way the expected application completion times assist traditional schedulers. However, defining heuristics to discover a standard behavior of the application checkpointing frequencies is challenging. One possibility is the user to initially supply the checkpoint frequency, and according to the executions, the scheduler checkpoint management service could refine the frequency value. Nevertheless, the frequency value would only make sense for a particular set of machines. Thus, such a value should be readjusted each time the application migrates to another set of machines.

Another interesting issue is on checkpoint granularity. It could be useful if the checkpoint management service could dynamically modify the granularity in which applications store their checkpoints. With this feature the scheduler could request this service to increase the frequency in which an application is generating the checkpoints, since it has the information that the resources used by such an application will not be available for a long time.

3.3 Cost Definitions

Naturally, migrating, shrinking and expanding the applications, as well as modifying the checkpoint granularity brings costs that should be taken into account. For this reason, a service to provide information about costs is important.

Changing the set of machines used by an application implies a lot of communication to redistribute the data and to balance the load depending on the new number of machines and their location. Unfortunately these communication costs are heavily dependent on the applications and the target platform. In previous work [6], we considered folding applications on specific numbers of machines to avoid complicated data redistribution. In this scope, applications have a natural number of machines n and can be set to execute on a fraction of this number of processors $\lceil n/i \rceil$ where i is an integer. With this restriction, the data is split in n groups at compile time and the communications always concern complete groups.

Therefore the delay induced in the schedule by a migration can sometimes be modeled and predicted to increase the overall efficiency of the schedules. This prediction is unfortunately not always possible and schedulers should thus try to minimize the number of migrations.

3.4 Inter Scheduler Communication

Inter scheduler communication is a service to allow user requests to be transfered among schedulers, thus increasing the system scalability. This is essential in distributed environments, such as Grids, where computer resources or services may not be available in a single domain. Here we illustrate two policies to be provided by this service: (i) local domain has preference and (ii) all domains are equivalents (Figure 2).

In the first policy a user submits an application to a known-scheduler, Scheduler 1 for example (Step 1), then the scheduler verifies whether the resources of its domain are suitable to execute the application or not (Step 2). In this verification the scheduler considers the current and future available resources - it could also verify if there will be enough time to generate the next checkpoint. Depending on the case, the scheduler then collects information from the other schedulers. Suppose Scheduler 1 discovers that Scheduler 4 has *better* resources to execute the application; Scheduler 1 then migrates the user request to Scheduler 4. Note that the meaning of *better* depends on a criterion established by the domain administrator or by the users themselves. Thus, *better* can be resources that either provide higher computer power or are more stable - lower probability to become unavailable.

In the second policy where all domains have the same preference, in Step 2, instead of checking if its own domain holds appropriate resources, Scheduler 1 requests information from all other schedulers, or from most of them, and based on a determined criterion it decides the *best* domain to execute the application.

An important issue when migrating user requests is the tradeoff between transparency and performance. In the example of the first policy, after Scheduler 1 forwards the user request to Scheduler 4, such a user may desire to monitor the execution, or the execution results are in order of Giga or Terabytes. Thus, users could request information and results from their executions through Scheduler 1, which will serve as a router between the user and Scheduler 4. On the other hand, Scheduler 1, after forwarding the user request to Scheduler 4, should communicate to user that it has no control about the request anymore, and all communication should be directly performed by Scheduler 4. Hence, in this second case the performance increases by decreasing the transparency.

Remark that the scheduler interconnection structure to improve the scalability can be implemented using different approaches. The example of Figure 2 illustrates a peer-to-peer structure. However, other approaches, such as hierarchies of schedulers, could also be utilized.

4 Scheduling Strategies

When we are dealing with complex distributed environments, such as Opportunistic Grids, we have to consider several scheduling problems. For example, (i) the amount of

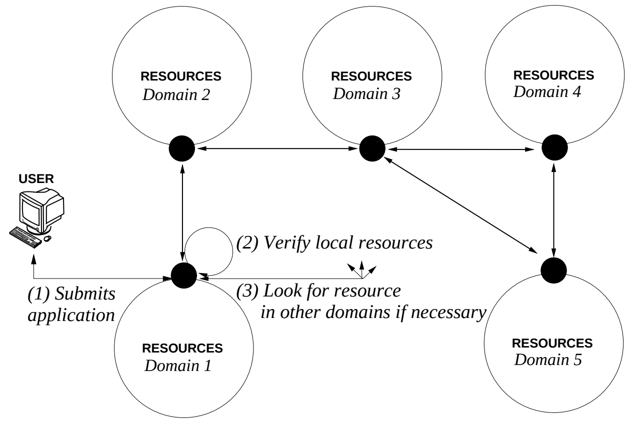

Figure 2: Multiple schedulers coordenate themselves to best select a set of resources to execute the user application.

applications to schedule is unknown, that is, applications are dynamically submitted to the scheduler; (ii) the amount of machines may vary; (iii) these machines may be heterogeneous; and (iv) the expected application execution times may be unavailable.

Furthermore, depending on issues such as administrative policies, user and application profiles, resource properties and services available, the scheduler can make use of different strategies, with different complexity levels, to assign application to resources. As each Grid domain can have its own environment settings, a definitive scheduler is not the most suitable choice. Therefore, flexibility is a key feature in a scheduler aimed at executing applications on Computational Grids. In this context, the proposed architecture aims to be flexible in the sense that scheduling strategies can be developed based on the available set of services, as those described in Section 3.

For instance, regarding the scheduling modes [14], a simple, or more sophisticated scheduling strategy can be employed. A simple strategy is the batch mode where the scheduler groups the applications and periodically, or based on some external event (such as resource availability), assigns them to resources. A more sophisticated strategy is the online mode, where the scheduler tries to assign applications to resources as soon it receives them. The advantage of batch mode is that the scheduler can consider more applications to make better scheduling decisions. Nevertheless, if a large number of applications is required to make these decisions, the delay to schedule may be high, and this may be a problem if we consider Quality-of-Service issues. If the batch mode is set up to schedule applications when a certain number of resources is available, there can be too many calls to the scheduler, or too few depending on the submitted applications. Thus the online mode is more appropriate on these conditions.

A scheduler usually performs decisions when an event occurs. Examples of events are: a machine joined or left the network; a user submitted an application or an application has just finished; a checkpoint is generated; or also a priority of an application is modified. Some of these events depend on services that assist the scheduler, and thus they are responsible for reporting the scheduler when they have a new information.

Suppose a scenario where an ongoing application generates a checkpoint and the checkpoint manager reports this event to the scheduler. Based on this event the scheduler could make decisions; for example: (i) the application should be interrupted since another application with higher priority requires its resources; (ii) based on resource prediction data, the application should not be executed further since there will not be enough time to generate the next checkpoint; (iii) the number of machines used by this application should be reduced due to resource prediction data; or (iv) the application should wait since new resources could be available in a short period of time.

Scheduling strategies could also consider application types as an additional criterion to take decisions. For example, suppose the scheduler should interrupt the execution of an application and start another one. The scheduler should rely on some criterion to chose which application must be started. Criteria such as submission time or application priority are generally used. However, the scheduler could also explore application types. In this example, thus, the scheduler could firstly attempt to schedule tightly-coupled applications, secondly the master-worker and PSAs. If none of them is possible, the scheduler could then try sequential applications. The order established in this example is based by the difficulty of each application type to deal with the loss of resources.

Another issue regarding application types is the period when they are executed. If applications do not support checkpointing, it is interesting to start the execution of tightly-coupled parallel applications at periods when resources will be available for a long time, at night or weekends, letting parameter sweep and master-worker applications to be executed at the other periods. Furthermore, in master-worker applications, good strategies for assigning machines to master processes is essential, since if the master process is suspended, the worker processes may be suspended as well. Thus it may be worth providing the scheduler with a personalized service to *best* select master machines.

Heterogeneity is also a common problem when scheduling applications on distributed environments. If applications execute on different machine types, load balancing algorithms must be taken into account. These algorithms could be directly implemented on the application or the scheduler could provide a service to deal with such a problem. Parameter sweep and master-worker applications are usually more suitable to work on heterogeneous machines by considering tightly-coupled applications. That is because tightly-coupled applications require communication among the processes. Thus, the delay of one process causes the delay of the overall application. A simple approach to minimize this problem with tightly-coupled applications is by grouping the machines in classes, and assigning them to only one machine class. Note that the scheduler could also be informed that the application is able to deal with heterogeneous machines.

5 Case Study: InteGrade

The InteGrade[1] project is a multi-university effort to build a novel Grid Computing middleware infrastructure aimed at leveraging idle computing power of desktop machines [10]. It supports the execution of sequential, parameter sweep and parallel applications implemented using the BSP model [5]. InteGrade relies on several components to provide different services, including resource prediction and management of checkpoint repositories. Figure 3 illustrates the InteGrade's architecture and its main components on a local network.

The two main components of the middleware core are Global Resource Manager (GRM) and Local Resource Manager (LRM). GRM is responsible basically for scheduling requests on a local network, while LRM aims at managing resources of the machine that provides processing power. In order to store user application information and binaries InteGrade relies on an Application Repository (AR). Local Usage Pattern Analyzer (LUPA) and Global Usage Pattern Analyzer (GUPA) are the components responsible for determining resource usage patterns. Each cluster machine executes LUPA to collect data about its local user usage patterns. Based on long series of data, it thus can derive usage patterns for that machine. GUPA is responsible for managing the information collected by LUPA from all network machines. Execution Manager (EM) is the component that is notified when an application is interrupted unexpectedly in order to assist the application restarting in a future through the use of checkpoint components. This interruption can be due to an execution cancellation, which is reported by LRM, or due to a machine unavailability, which is reported by GRM.

InteGrade also provides a checkpointing library to enable portable checkpointing mechanism for sequential, parameter sweep and BSP parallel applications [5]. Finally, Checkpoint Repository Manager (CRM) is responsible for coordinating local checkpoint repositories in order to allow a local process to access checkpoint files from other machines. When it is necessary to store a checkpoint, the checkpointing library queries CRM requesting the available checkpoint repositories in order to transfer them the checkpoint data.

As we can observe, InteGrade provides two main services to assist in the scheduling decisions, which are the resource prediction and the checkpointing support. Resource prediction service is a direct use of the LUPA and GUPA components. However, for checkpointing management service, as the InteGrade's components that provide checkpointing support are scattered, they cannot be straightforwardly used by the scheduler. A checkpoint management service could be developed to make use of the available related components. For example, each time CRM is requested to supply the list of checkpoint repositories, CRM could notify the scheduler that an application has recently generated a checkpoint, and thus one of the strategies described in Section 4 could be used.

[1]Project homepage: http://integrade.incubadora.fapesp.br

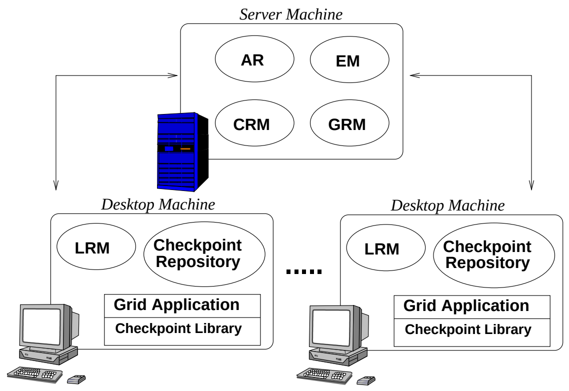

Figure 3: InteGrade's architecture and its main components.

Considering the checkpoints generation frequency in BSP parallel applications, in the current InteGrade's version a fixed time is established to generate checkpoints, which are performed by the coordinator process - similar to the master process in master-worker applications. Thus, basically the checkpoint service should allow the scheduler to dynamically configure the checkpointing frequency. Furthermore, currently the Execution Manager is the only component that holds the information on which machine is executing the coordinator process. Therefore, a method to supply this information to the checkpoint service should also be implemented.

6 Conclusion and Further Work

Scheduling different application classes on distributed networks is challenging, in particular when machines that comprise such an environment are non-dedicated. In this paper we have proposed a flexible architecture for scheduling parallel applications on opportunistic networks which relies on services that are used in the form of plug-ins. It is important to mention that a flexible architecture is a promising approach in the context of Computational Grids since each Grid domain can attend its own needs such as administrative policies, user and application profiles and resource properties. Furthermore, the scheduler can be extended by the inclusion of new, and even currently unknown services.

In this paper we have described some support and advanced services. Fault tolerance, resource prediction, and monitoring are examples of essential services for the scheduling of parallel applications on non-dedicated machines. Checkpointing management is a desirable service to reduce wasted work generated due to the unavailability of a machine. This service is specially important when we consider tightly-coupled parallel applications, where an interruption of a process can crash the overall application. Exploring techniques to dynamically modify both the checkpointing frequency and amount of resources is also interesting for a better usage of the available resources. Also the interconnection among schedulers is a useful service to improve the system scalability, as well as the cooperation among the different administrative domains that comprise the Grid.

As future work we intend to implement a prototype of the proposed scheduler using the InteGrade Grid middleware as a testbed environment. This first prototype will mainly explore the InteGrade's checkpointing components. Moreover we intend to investigate the common service interfaces of the existing Grid middleware systems. With this study we expect to establish a set of easy-to-use interfaces to promote the use of the proposed scheduler on several Grid middleware systems, specially those aimed at working with opportunistic networks.

References

[1] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[2] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive Computing on the Grid Using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):369–382, 2003.

[3] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the 9th Heterogeneous Computing Workshop*, page 349, Washington, DC, USA, 2000. IEEE Computer Society.

[4] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5):597–610, 2003.

[5] R. Y. de Camargo, A. Goldchleger, F. Kon, and A. Goldman. Checkpointing-based rollback recovery for parallel applications on the integrade grid middleware. In *Proceedings of the 2nd workshop on Middleware for grid computing*, pages 35–40, New York, NY, USA, 2004. ACM Press.

[6] P.-F. Dutot, M. A. S. Netto, A. Goldman, and F. Kon. Scheduling moldable BSP tasks. In *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2005)*, Lecture Notes in Computer Science, pages 157–172, Cambridge, MA, 2005. Springer.

[7] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, 2002.

[8] D. G. Feitelson and L. Rudolph. Towards convergence in job schedulers for parallel supercomputers. In *IPPS '96: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–26, London, UK, 1996. Springer-Verlag.

[9] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufman, San Francisco, CA, 1999.

[10] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G. C. Bezerra. InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. *Concurrency and Computation: Practice and Experience,* 16(5):449–459, March 2004.

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.

[12] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Adaptive scheduling for master-worker applications on the computational grid. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing (GRID 2000)*, Bangalore, India, December 2000.

[13] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Evaluation of strategies to reduce the impact of machine reclaim in cycle-stealing environments. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 320, Washington, DC, USA, 2001. IEEE Computer Society.

[14] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002.

[15] J. Liang, K. Nahrstedt, and Y. Zhou. Adaptive multi-resource prediction in distributed resource sharing environment. In *Proceedings of 4th IEEE International Symposium the Cluster Computing and the Grid*, pages 293–300, 2004.

[16] V. S. Pande, I. Baker, J. Chapman, S. Elmer, S. M. Larson, Y. M. Rhee, M. R. Shirts, C. D. Snow, E. J. Sorin, and B. Zagrovic. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Peter Kollman Memorial Issue, Biopolymers*, 68(1):91–109, 2003.

[17] J. Pruyne and M. Livny. Interfacing condor and pvm to harness the cycles of workstation clusters. *Future Generation Computer Systems*, 12(1):67–85, 1996.

[18] V. S. Sunderam. PVM: a framework for parallel distributed computing. *Concurrency, Practice and Experience*, 2(4):315–340, 1990.

[19] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[20] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[21] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, 1999.

[22] M. Wu and X.-H. Sun. A general self-adaptive task scheduling system for non-dedicated heterogeneous computing. In *Proceedings of the IEEE International Conference on Cluster Computing*, page 354, Hong Kong, China, 2003. IEEE Computer Society.