OGST: An Opportunistic Grid Simulation Tool

Gilberto Cunha Filho
Universidade Federal do Maranhão
Programa de Pós-Graduação
Departamento de Engenharia de Eletricidade
São Luís, MA, Brazil
gilberto.cunha@gmail.com

Francisco José da Silva e Silva
Universidade Federal do Maranhão
Departamento de Informática
São Luís, MA, Brazil
fssilva@deinf.ufma.br

Abstract

Simulation tools play a fundamental role on the development of Grid middlewares since they provide a controllable and low cost environment for validating new concepts and implementations that address aspects such as heterogeneity, scalability, security and the dynamism of Grid environments.

This paper describes the motivation, design principles, architecture, and implementation of the Opportunistic Grid Simulation Tool - OGST, an object-oriented discrete event simulator whose main objective is to assist developers of opportunistic Grid middlewares on validating new concepts and implementations under different execution environment conditions and scenarios. The preliminary motivation for OGST development was to provide a way for evaluating the behavior of adaptive and autonomic mechanisms for opportunist Grids, such as adaptive scheduling approaches and dynamic re-scheduling of applications.

1 Introduction

A computer grid comprises a hardware and software infra-structure that allows integration and sharing of distributed resources, such as software, data and peripherals, inside and among institutions. Computational grids have become an attractive alternative for execution of applications that demand huge computational power and have been used to solve problems in varied areas of scientific, enterprise, and industrial activities, such as: computational biology, image processing for medical diagnosis, weather forecast, high energy physics, marketing simulations, and oil prospection.

Since its first developments in the early to mid 1990s, several research groups have been addressing the complexity of building the Grid software infrastructure, facing challenges such as the support for huge resource heterogeneity; high scalability of distributed resources; efficient resource allocation and management; dynamic resource scheduling, transparent code mobility, in order to promote load balancing and support for non-dedicated nodes; distributed fault-tolerance; security services and policies that must spread through different administrative domains, and so on.

During the development of Grid middlewares, researchers often employ simulation tools and techniques for validating new concepts and implementations. Simulation tools play a fundamental role on the development of Grid middlewares since: (a) researchers often do not have access to huge Grid testbed environments, limiting the capacity for evaluating situations that demand high amount of resources; (b) it is difficult to explore in large scale application and resources scenarios involving several users in a repetitive and controlled way, due to the dynamic nature of Grid environments; (c) real Grid applications usually consume great amount of time, ranging from a few hours to even weeks.

During the last four years, our research group has taken part on the development of the InteGrade project[1] [12], a multi-university effort to build a novel grid computing middleware infrastructure to leverage the idle computing power of personal workstations for the execution of computationally-intensive parallel applications. Recognizing that the dynamic nature of the grid infrastructure, its high scalability, and great heterogeneity has turn impracticable its configuration, maintenance, and recovery in case of failures solely by human beings, we are currently involved on developing autonomic mechanisms that can be applied on Grid middlewares [18]. Regarding self-optimization, we are investigating adaptive scheduling approaches and dynamic re-scheduling of applications on InteGrade. In order to be considered successful, this work must provide answers to fundamental questions, such as: is it worth

[1]Homepage: `www.integrade.org.br`

to perform dynamic adaptations to the Grid scheduling policy? What are the costs/benefits involved? When adaptive actions should be applied? What are the adaptive actions that should be considered? For finding answers to those questions, we are making extensive use of simulations, evaluating the behavior of several scheduling algorithms commonly used on Grid environments on different execution environments conditions. We are also investigating how parameters used on those algorithms can be dynamically adjusted as the execution environment changes.

This paper describes the motivation, design principles, architecture, and implementation of the Opportunistic Grid Simulation Tool - OGST, a simulator we developed for evaluating adaptive scheduling approaches for opportunistic Grid middlewares. It also presents the results and conclusions obtained with a first set of simulations performed. This paper is organized as follows: Section 2 discusses opportunistic Grids concepts and gives an overview of the InteGrade middleware; Section 3 presents the requirements, architecture, and implementation of OGST; Section 4 presents simulations performed and their analysis; Section 5 presents a brief overview of other Grid simulation tools, highlighting the differences from our approach, while on Section 6 we drove the conclusions of our work and discuss some future steps of our research.

2 Opportunistic Grids and the InteGrade Middleware

Currently private and public institutions have a large number of computing resources, such as personal computers and workstations, with great capacity for data processing and storage. Computers are idle most of the time and, even when they are in use, usually only a low percentage of their computing capacity is effectively used [17]. Opportunistic Grids are computing systems that provide the means for using an installed base of regular computers for executing high performance computing applications, leveraging the available idle computing power [12]. The focus of an opportunistic Grid middleware is not on the integration of dedicated computer clusters (e.g. Beowulf) or supercomputing resources, but on taking advantage of idle computing cycles of regular computers and workstations that can be spread through several administrative domains.

Developers of an opportunistic Grid middleware must deal with several challenges, such as: (a) great instability, since nodes are usually not dedicated and applications do not execute on a controlled environment; (b) high heterogeneity of computing resources and network links, since computers are usually spread through different administrative domains; (c) the middleware should not interfere on the regular use of computing resources or, at least, should provide minimum impact on the performance perceived by its users, otherwise it will be difficult to obtain their consent for resource usage; (d) it is desirable the provision of mechanisms to predict the time-span on which a machine will be idle in the future, minimizing the need for code migration.

2.1 InteGrade Overview

The InteGrade middleware is a fully object oriented system that provides a robust and flexible software infrastructure for opportunistic Grid computing. An InteGrade Grid basic architectural unit is a cluster, a collection of machines usually connected by a local network. Clusters can be organized in a hierarchy, allowing to encompass a large number of machines. Each cluster contains a *Cluster Manager* node that executes InteGrade components responsible for managing the cluster computing resources and for inter-cluster communication. Other cluster nodes are called *Workstations*, which export part of its resources to Grid users. They can be shared or dedicated machines.

InteGrade currently allows the execution of three application classes: (a) regular applications, where the executable code is assigned to a single Grid node; (b) parametric or BoT (Bag of Tasks) applications, where several copies of the executable code (tasks) are assigned to different Grid nodes and each of them processes a subset of the input data independently and without exchanging data; (c) parallel applications following the BSP or MPI models, whose processes occasionally exchange data between themselves.

3 OGST

OGST (Opportunistic Grid Simulation Tool) is a simulation utility whose main objective is to assist developers of opportunistic Grid middlewares on validating new concepts and implementations under different execution environment conditions and scenarios. The preliminary motivation for OGST development was to provide a way for evaluating the behavior of scheduling algorithms commonly used on Grid environments under different execution environment conditions and the investigation of adaptive scheduling approaches and dynamic re-scheduling of applications.

Considering the simulation engine taxonomy described by Sulistio et al [19], OGST is an object-oriented discrete event simulator (DES). OGST allows the simulation of large scale applications and resources scenarios involving several users in a repetitive and controlled way. It was build on top of the GridSim [4] simulation toolkit.

We defined OGST design principles in accordance with the characteristics of opportunistic Grids, establishing the following requirements for its development: (1) it must provide support for defining Grid environments that exhibits high heterogeneity of machines and network links; (2) it must allow the definition of heterogeneous applications, ranging from regular to bag-of-tasks and parallel applications, that consume great amount of time, varying from a few hours to even weeks; (3) it must allow the simulation of frequent join and leave of nodes, since nodes are usually not dedicated on opportunistic Grids; (4) it must allow node and link failure injection, due to the common instability of opportunistic Grids; (5) it must provide support for simulating application fault tolerance mechanisms commonly applied on opportunistic Grids (e.g. restarting, checkpointing, and replication [10]) in order to ensure the application execution progress even in an unstable execution environment; (6) it must allow the simulation of variant availability of each Grid node, considering different usage periods; (7) it must allow the scheduling algorithm to obtain several informations concerning the applications and the execution environment, such as the estimated task completion time and input files size and Grid nodes processing capacity and load, in order to ease the development of a wide range of scheduling algorithms; (8) it must allow the dynamic replacement of the scheduling algorithm and/or dynamic adjustment of scheduling parameters during the simulation in order to allow the evaluation of adaptive scheduling approaches; (9) it must allow the use of traces collected from real environments (such as node availability) in addition to synthetic data; (10) it must provide the storage of relevant simulation data, such as applications submission and conclusion timestamps.

3.1 OGST Architecture

We developed OGST architecture as an object-oriented system. It was developed in the context of the InteGrade project, but was designed to allow the simulation of generic opportunistic Grids in order to be applied by other Grid middleware research projects.

Figure 1 illustrates OGST main components. The `Feature Generator` (FG) is a utility used for defining the simulated Grid environment (nodes and network links) and applications with their arrival rate. OGST currently allows the simulation of regular and bag-of-tasks applications. For each application task, its length (in million instructions) must be provided. The `Application Submission and Control Tool` (ASCT) represents the Grid user and is responsible for application submission, receiving notification about its conclusion.

The `Global Resource Manager` (GRM) receives application submissions from the ASCT and runs the `Scheduling Strategy` (SS). The scheduling algorithm uses data about the availability of Grid resources provided by the `Trader Manager` (TM). Each application task is then mapped for execution on a specific

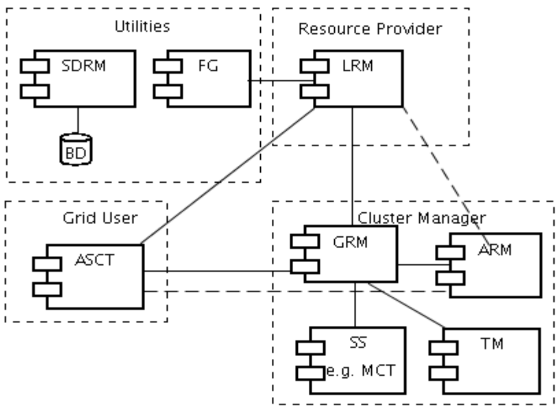

Figure 1. OGST Main Components

Grid node. Each Grid node runs a `Local Resource Manager` (LRM), responsible for the instantiation and execution of application tasks scheduled to the node, maintaining a list of tasks waiting for execution. It is also responsible for varying the local resource load. The `SimulationDataRecordManager` (SDRM) uses a relational database for storing the simulation collected data, such as each task start and conclusion timestamps. The TM component is also responsible for simulating node failure and recovery. OGST allows the simulation of application execution replication, commonly used on opportunistic Grid environments in order to circumvent eventual node failures. The `Application Replication Manager` (ARM) is responsible replica management.

3.2 OGST Implementation Highlights

OGST was written in Java and is an extension of the GridSim [4] toolkit which, by its turn, inherits event management and threaded entity features from SimJava [14], adding to this simulator networking and event delivery features that allows synchronous and asynchronous communication for service access and delivery. GridSim provides a base class called `GridSim` that provides the communication between OGST components based on an event-driven approach. This subsection describes the extensions to GridSim provided by OGST.

OGST allows the automatic creation of simulated Grid environments composed of a large amount of highly heterogeneous nodes. This feature is provided by the `generateMachines()` method of the `Feature Generator` utility, which receives as input parameters the path where a text file containing the description of the generated machines should be stored, the desired amount of nodes, and the processing capacity of the slowest and fastest Grid nodes. Nodes are generated according to an uniform distribution. A node processing capacity is defined in MIPS (Millions Instruction Per Second) as per SPEC (Standard Performance Evaluation Corporation) CPU (INT) 2000 benchmark rating[2].

GridSim does not explicitly define an application execution model. The provided examples assume that all tasks of a given application execute on a single parallel machine at a user provided arrival time. OGST explicitly defines three application execution models: regular, bag-of-tasks, and parallel, using the same semantics presented on Section 2. Our current implementation allows the definition of only regular and bag-of-tasks applications. OGST provides the automatic generation of synthetic applications, a feature provided by the `generateRegularApp()` and `generateBoTApp()` methods of the *Feature Generator* utility. The first method receives as input parameters the path where a text file containing the description of the generated applications should be stored, the desired amount of applications, the desired application arrival rate per minute, and the minor and major application length to be generated. The generated applications length, defined in MI (Million Instructions), follows an uniform distribution. The `generateBoTApp()` receives the same first three parameters, a value that defines the amount of tasks to be generated for each application, the average task length, and a value that defines a variation percentage based on the provided average task length, allowing the generation of tasks with different lengths in order to simulate task heterogeneity. The generated tasks lengths follows an uniform distribution while the tasks arrival time follows a Poisson distribution, since the superposition of a large number of independent renewal processes is approximately a Poisson process, as stated by the Palm-Khintchine theorem [13].

GridSim failure injection architecture assumes that the average amount of resources that should fail must be defined at the beginning of the simulation [5]. The amount of failing nodes, their failure time, and the failure duration time follow a hyper-exponential distribution and are generated before the simulation start. Since our objective is to evaluate scheduling heuristics under different execution environment conditions (such as different failure rates), we alter this behavior by allowing the definition of a desired simulation mean time between node failures. Node failure and its recovery are, then, generated during the simulation execution and follow a exponential distribution.

Concerning fault tolerance mechanisms, GridSim allows the simulation of an automatic restart of failed tasks. OGST extends GridSim fault tolerance mechanism by providing support for simulating the use of checkpoint and replication, other common techniques used on opportunistic Grids. The simulated checkpoint mechanism is based on a globally available distributed stable storage (as provided on InteGrade [8]), allowing a fast task restart on another Grid machine in case of node failure. OGST also allows the simulation of task migration, necessary when a user requests the use of a Grid resource, since regular non-dedicated machines are used.

GridSim provides the necessary support for implementing scheduling heuristics, delegating this task to the user. OGST provides a library of scheduling algorithms currently composed of four scheduling heuristics: InteGrade, OLB, MCT, and Min-min [16, 3]. A simulation scheduling strategy is currently instantiated by the `SS` class, using the strategy pattern [11]. On-going work is in curse for providing support for dynamic replacement of scheduling algorithms.

GridSim can collect for each simulation execution statistical data that are stored on a text file, which can be used for generating an execution report. However, querying and finding data relations on a large amount of statistical data by manipulating a text file can become an arduous task. For simplifying the query and analysis of a large amount of generated data, OGST `SDRM` uses a relational database. `SDRM` also allows the automatic generation of graphs from data collected through several simulation executions, such as the average application completion time as a function of the mean time between node failures or as a function of the arrival rates. These graphs are generated using the gnuplot tool [3].

Since an opportunistic Grid make use of regular non-dedicated machines, its execution environment exhibits a variant availability of each Grid node, considering different usage periods. To simulate this usual behavior, GridSim allows the definition of the local workload for each resource according to peak hour, off-peak hour, weekends, and holidays. We are currently working on the support for allowing the definition of resources work load based on trace files.

4 Simulations

Scheduling on opportunistic Grids is a challenging problem due to several factors, such as the existence of a non predetermined and dynamic resource pool, high interconnection and node heterogeneity, high scalability, and a non-controllable environment composed of non-dedicated personal computing nodes. These characteristics justify the importance of investigating efficient application scheduling heuristics that takes into consideration aspects such as fault tolerance, migration, information management, load balancing, adaptive scheduling, and application re-scheduling.

This Section describes the simulations we performed using OGST with the aim of analysing the performance of different scheduling algorithms (MCT, Min-min, OLB, InteGrade) under different execution environment conditions,

[2]`http://www.spec.org/osg/cpu2000/results/res2003q2/cpu2000-20030422-02132.html`

[3]Homepage:`http://www.gnuplot.info`

considering the objective of minimizing the applications average completion time.

The simulated Grid environment was composed of 100 machines with an average processing power equivalent to a Pentium 4 with 2.8 GHz (1000 MIPS), which was considered a representative value for regular personal computers. In order to take into consideration the environment heterogeneity, Grid nodes were generated according to an uniform distribution $U(222, 1776)$ MIPS, having the fastest machine a processing power 8 times higher then the slowest one.

We simulated the execution of 1500 regular applications, synthetically generated with a variant length (in millions of instructions) through an uniform distribution $U(3015\times10^4, 30150 \times 10^4)$ MI, which takes approximately 5 to 50 hours of execution on an Pentium 4 with 2.8 GHz.

The simulations took into consideration the variability of two execution environment parameters: the applications arrival rate and the mean time between node failures. Simulations were performed for each scheduling algorithm being evaluated considering the following arrival rates per minute: 0.025 (low) and 0.25 (high). The applications arrival time were generated according to a Poisson distribution. For each arrival rate, we also observed each scheduling algorithm behavior considering the following mean time between node failures: 1, 2, 4, 6, 8, 10, 12, and 14 hours. Failures were generated according to an exponential distribution and node recovery followed an uniform distribution $U(6, 48)$ hours, in order to characterize short and long fails, ranging from network failures to a machine crash.

For each combination of scheduling algorithm, arrival rate and mean time between node failures previously described, we also took into consideration the use of two different fault tolerance mechanisms: restart and checkpointing, leading to a total of 128 different simulations that were performed on an Intel Pentium 4 2.8 GHz with 2 GB RAM running the Linux kernel version 2.6.24-19. Each simulation was repeated 30 times, resulting on 3840 experiments. For each experiment, we measured the completion time per application and calculated the average completion time of the 1500 applications. We, then, calculated the average of the average application completion time for each of the 30 applications set that, for the reminder of this Section, will be called just as the average completion time per application.

Figure 2 shows the average completion time per application as a function of the mean time between failures when the applications arrival rate was 0.025 per minute and applications were just restarted for execution on another node in case of a node failure. With a short interval between failures (1 hour), OLB achieved the best performance (lower average completion time per application) since it only schedules one task per machine reducing the required number of tasks resubmissions in case of a node failure, an advantage when the environment exhibits a high failure rate.

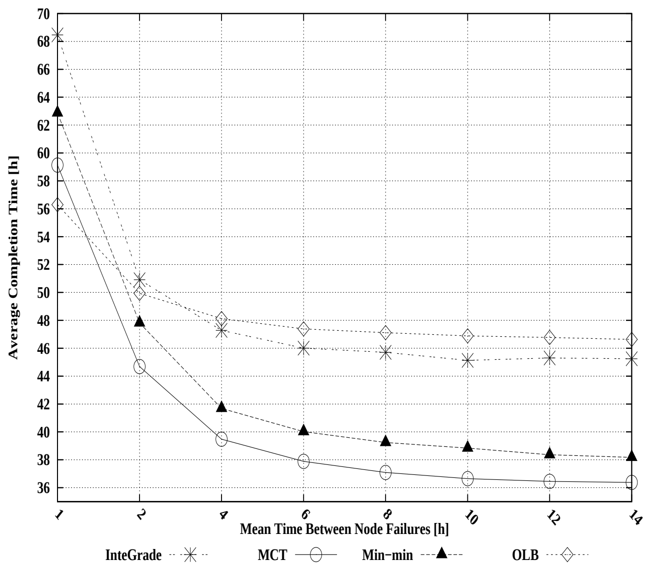

Figure 2. Average application completion time with restart and an arrival rate of 0.025 applications per minute.

As the failure rate becomes lower (2 to 14 hours of mean time between node failures), MCT and Min-min performed better than OLB, since they can perform a better task mapping by using information about the estimated application completion time. MCT performed better than Min-min since in this simulation set the application arrival rate was low (0.025 per minute) and MCT uses an on-line approach, scheduling the tasks as soon as they arrive to the scheduler, while Min-min uses a batch approach, which collects a set of tasks and perform their mapping only at prescheduled interval times called a mapping event. On our simulations, Min-min uses a 2 minutes interval time between mapping events.

Considering the same application arrival rate of 0.025 per minute, we also evaluated the scheduling heuristics when checkpoint is used for restarting failed applications from their last saved state. In this case, even when the failure rate is high (one hour interval between failures), MCT and Min-min performed better than OLB, since the lower number of tasks resubmissions obtained with OLB becomes less relevant, due to the fact that the tasks do not need to be restarted from scratch.

Figure 3 shows the simulations results when the environment exhibits a high arrival rate (0.25 applications per minute) and checkpoint is used for task recovery. In this case, Min-min obtained the best results, since the higher number of tasks maintains the Grid resources busy and more applications compose the mapping event set, allowing the comparison of their estimated completion time, which leads to a better task mapping. Using task restarting instead of checkpointing, the performance ordering of the evaluated algorithms is the same, with Min-min obtaining the best results, emphasizing the better performance of batch algorithms when the Grid resources are stressed.

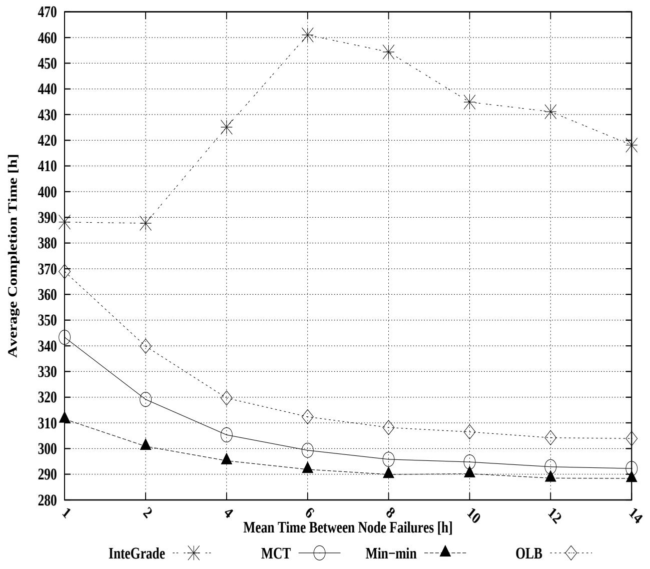

Figure 3. Average application completion time with checkpointing and an arrival rate of 0.25 applications per minute.

This first set of simulations highlighted that scheduling heuristics perform differently under diverse execution environment conditions and gave preliminary answers about when one approach is better than the other and how much can be gain by switching the scheduling heuristic, strengthened the benefits that can be achieved by applying an adaptive scheduling approach.

5 Related Work

Tools for simulating Grid and volunteer computing are emerging as they provide a controllable and low cost environment for evaluating new architectures and approaches that copes with the challenges of providing a reliable, highly distributed, and large scale environment running computer intensive long term applications.

SimBOINC[4] [15] is a simulator for heterogeneous and volatile desktop Grid and volunteer computing systems based on SimGrid [6]. As BOINC, the simulator follows a client/server model that runs bag-of-tasks applications, whose tasks execute independently, without exchanging data between themselves. The simulator uses a pull approach, where clients request work from servers, execute the task and return the results. SimBOINC was used to evaluate scheduling heuristics that runs locally on each client machine [1]. It also simulates a local checkpoint mechanism that allows tasks to be automatically restarted when a failed machine returns to work. OGST is based on the push model, where the scheduler is responsible for mapping tasks to resource providers, but it also allows the implementation of local scheduling heuristics and application re-scheduling approaches. Regarding fault-tolerance, OGST was designed to provide three mechanisms: restart, checkpoint, and replication. The simulated checkpoint mechanism is based on a globally available distributed stable storage (as provided on InteGrade [9]), allowing a fast task restart on another Grid machine in case of node failure. OGST also allows the simulation of task migration, necessary when a user requests the use of a Grid resource, since regular non-dedicated machines are used.

SimGrid [6] is a simulation framework written in C, that provides a generic evaluation tool for large-scale distributed computing. Like GridSim, it allows the evaluation of cluster, Grid, and P2P algorithms through components that model and simulate Grid resources, applications, schedulers, and users. It provides a scalable and extensible trace-driven simulation engine that allows the simulation of arbitrary network topologies, resource availability, and failures. OGST, on the other hand, focus on providing support for simulating the dynamics of opportunistic Grids, modelling the frequent join and leave of nodes, different application models (regular, bag-of-tasks, and parallel), different fault-tolerance mechanisms, and code migration. OGST is event-driven and on-going work is in course for providing support for trace-driven.

OptorSim [2] allows the simulation of Data Grid environments, where very large data sets are processed in a distributed way. The simulator architecture is based on the EU DataGrid[5]. OptorSim objective is to investigate the stability and transient behavior of replication optimization methods. It focus on data replication, which evolves replica creation and management in different geographical locations and methods for minimizing data access cost. The simulator was used to compare several replication strategies, such as LRU-based strategies and economic based approach.

6 Conclusion and Future Work

This paper described OGST, an object-oriented discrete event simulator whose main objective is to assist developers of opportunistic Grid middlewares on validating new con-

[4]Homepage:`http://simboinc.gforge.inria.fr`

[5]Homepage:`http://www.edg.org`

cepts and implementations under different execution environment conditions and scenarios. It was developed in the context of the InteGrade project, but was designed to allow the simulation of generic opportunistic Grids in order to be applied by other Grid middleware research projects.

OGST was carefully designed to take into consideration the dynamics of opportunistic Grids, providing a set of features that hasten the development of simulations that takes into consideration the dynamism of the execution environment. Since the preliminary motivation for OGST development was to provide a way for evaluating the behavior of scheduling algorithms commonly used on Grid environments under different execution environment conditions and the investigation of adaptive scheduling approaches, this paper also described a first set of simulations performed that strengthened the benefits that can be achieved by switching from different scheduling heuristics considering a variant execution environment.

Ongoing work includes the use of traces that will allow the use of real data concerning resource availability, the support for dynamic replacement of scheduling heuristics, and the simulation of predictions of resource availability based on resource usage patterns, using as foundation the work described in [7].

7 Acknowledgments

This work, as part of the InteGrade project, is supported by the Brazilian Federal Research Agency, CNPq, grant No.55.0895/2007-8.

References

[1] D. Anderson. BOINC: A System for Public-Resource Computing and Storage. *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, 2004.

[2] W. Bell, D. Cameron, A. Millar, L. Capozza, K. Stockinger, and F. Zini. Optorsim: A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17(4):403, 2003.

[3] T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, et al. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[4] R. Buyya and M. Murshed. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[5] A. Caminero, A. Sulistio, B. Caminero, C. Carrion, and R. Buyya. Extending GridSim with an architecture for failure detection. *International Conference on Parallel and Distributed Systems*, 2:1–8, 2007.

[6] H. Casanova, A. Legrand, and M. Quinson. Simgrid: A generic framework for large-scale distributed experiments. *Tenth International Conference on Computer Modeling and Simulation*, pages 126–131, April 2008.

[7] D. Conde. Anlise de padrões de uso em grades computacionais. Master's thesis, Department of Computer Science - University of São Paulo, Brasil, SP, January 2008.

[8] R. Y. de Camargo, R. Cerqueira, and F. Kon. Strategies for storage of checkpointing data using non-dedicated repositories on grid systems. In *ACM/IFIP/USENIX 3rd International Workshop on Middleware for Grid Computing*, Grenoble, France, November 2005.

[9] R. Y. de Camargo and F. Kon. Design and implementation of a middleware for data storage in opportunistic grids. In *7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, Rio de Janeiro, Brazil, 2007.

[10] S. A. de Sousa, F. J. da Silva e Silva, and R. F. Lopes. A flexible fault-tolerance mechanism for the integrade grid middleware. In *ICNS '07: Proceedings of the Third International Conference on Networking and Services*, page 26, Washington, DC, USA, 2007. IEEE Computer Society.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: elements of reusable object-oriented software. *Addison-Wesley Professional Computing Series*, page 395, 1995.

[12] 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 & Experience. Vol. 16, pp. 449-459*, 2004.

[13] D. Heyman and M. Sobel. Stochastic models in operations research. Vol. I: Stochastic processes and operating characteristics. 1982.

[14] F. Howell and R. McNab. SimJava: A Discrete Event Simulation Package For Java With Applications In Computer Systems Modelling. *Proceedings of the First International Conference on Web-based Modelling and Simulation*, 1998.

[15] D. Kondo. Simboinc: A simulator for desktop grids and volunteer computing systems.

[16] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *HCW '99: Proceedings of the Eighth Heterogeneous Computing Workshop*, page 30, Washington, DC, USA, 1999. IEEE Computer Society.

[17] M. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Performance Evaluation*, 12(4):269–284, 1991.

[18] M. A. S. Sallem and S. A. de Sousa. Autogrid: Towards an autonomic grid middleware. In *WETICE '07: Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 223–228, Washington, DC, USA, 2007.

[19] A. Sulistio, C. Yeo, and R. Buyya. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Software- Practice and Experience*, 34(7), 2004.