DYNAMIC SCHEDULING OF COMPUTATIONALLY INTENSIVE APPLICATIONS ON UNRELIABLE INFRASTRUCTURES

Maria Chtepen, Bart Dhoedt,
Filip De Turck and Piet Demeester
Department of
Information Technology
(INTEC)
Ghent University
Sint-Pietersnieuwstraat 41,
B-9000 Gent, Belgium
Maria.Chtepen@intec.ugent.be

Filip H.A. Claeys and
Peter A. Vanrolleghem
Department of Applied
Mathematics, Biometrics and
Process Control (BIOMATH)
Ghent University
Coupure Links 653,
B-9000 Gent, Belgium
Filip.Claeys@biomath.ugent.be

KEYWORDS
Grid computing, resource availability, dynamic scheduling, simulation, software development

ABSTRACT

Grid is a distributed computing paradigm that has considerably gained importance during the last several years. Typical for grids is that their resources are highly distributed and often belong to different organizations. Therefore, resource availability and stability form an important issue that should be taken into account when designing scheduling mechanisms for grid systems. Currently, most existing grids make use of static schedulers, which is not sufficient to deal with the dynamic nature of grid resources. This paper is touching on the dynamic scheduling concept, justifying its usefulness for computationally intensive applications.
All results claimed in this paper are based upon simulations done in the DSiDE simulation environment. The DSiDE simulator is a discrete-event simulator for grids that differs from similar grid simulators, such as NSGrid, GridSim and SimGrid, on its ability to easily simulate the dynamics of grid jobs and resources, as well as on its short simulation times and portability.
As a use-case for this study, a computationally intensive generic modelling and simulation tool, named Tornado, was chosen. The effect of dynamic resource availability on the execution of Tornado jobs will be shown.

INTRODUCTION

Grid computing (Foster and Kesselman 1999) is a relatively new technology in the domain of distributed computing, which has recently gained importance as a consequence of a continuously increasing demand for elaborate sources of computational power. Grids are defined as an aggregation of heterogeneous, (globally) distributed resources, often belonging to different administrative domains. Although functionally different classes of grids exist (compute, information, desktop, *etc.*), most of them define the same set of services: through a User Interface users are able to submit their workload (jobs) to a grid system; a Scheduler assigns jobs received from UIs to distributed Computational and/or Storage Resources; Information Services, collects information about the grid status that helps the Scheduler to take decisions for job assignment.
The ultimate goal of grid technology is to allow for transparent execution of a user's jobs, optimally exploiting the combined capacities of multiple resources while taking into account administrative policies, user requirements and system status. The ability to accomplish this goal implies the existence of an intelligent and flexible scheduling mechanism for grids.
In general, scheduling algorithms can be divided into two broad categories: static and dynamic. Static algorithms assign jobs to available resources before the execution of the jobs starts. Once jobs are running, they can no longer be interrupted by the scheduler. In case of dynamic algorithms, previously taken scheduling decisions are regularly re-evaluated and adjusted to the changing status of grid resources and jobs. It is clear that dynamic scheduling is much more complex to accomplish than its static equivalent, therefore currently there exist very few systems supporting rescheduling (Roy and Livny 2003) (Barak and La'adan 1998) (Gehring and Reinefeld 1996).
In distributed environments, such as grids, with highly dynamic resources and diverse user requirements, situations where static scheduling does not suffice to guarantee efficient workload execution often occur. For example, resources can fail or become unavailable; new resources can join the system; load on computational or network resources can vary significantly; jobs with high priority or critical deadlines can arrive, requiring the best possible execution service. In grids that make use of cycle scavenging, jobs are typically submitted to a number of free machines, but during execution the load of the computational resources can change drastically. At that moment the initial assignment of the workload is no longer "optimal". In a system that employs static scheduling the jobs will continue executing on the slow machines while more appropriate resources can be unutilized. From this example it is clear that static methods are not suitable to guarantee the optimal resource utilization in highly dynamic systems.

The paper is structured as follows: in the next section existing grid simulation projects and the objectives for the development of the DSiDE simulation environment are discussed. A further section describes in more detail the design of DSiDE. Simulation experiments with varying jobs executed on an unreliable grid infrastructure and their results are presented in the section "Simulation results". A case study on Tornado follows this discussion. The paper is terminated with concluding remarks and an overview of future work.

DSIDE AND RELATED GRID SIMULATORS

Next to the broad spectrum of general-purpose discrete-event simulation environments, such as GPSS and SLX (http://www.wolverinesoftware.com), there are also a number of well-known grid simulators. Examples of the latter include GridSim, SimGrid and NSGrid.

GridSim (Buyya and Murshed 2002) is a grid simulator based on JavaSim. It provides high extensibility and portability through Java and thread technologies, and allows for modelling of the most common grid components, including rough network models. GridSim initiates each component in a modeled system as a distinct thread, with a separate event queue. Although very flexible, GridSim is not scalable since it depends on a number of threads that is limited. In addition, the thread management overhead in Java results in slow run times (Phatanapherom et al. 2003).

SimGrid (Legrand et al. 2003) is a C-based discrete-event job scheduling simulation library, suitable for modelling centralized as well as distributed workload scheduling strategies. It provides highly accurate TCP (Transfer Control Protocol) and non-TCP network models for simulation of data-intensive applications. SimGrid is much faster than most Java-based simulators. However, its approach of using user-level threads to model resources renders SimGrid limited by thread switching capability and introduces significant system overhead (Phatanapherom et al. 2003).

NSGrid (Thysebaert et al. 2004) is a grid simulation framework that accurately models network traffic. It implements grid components on top of the NS simulator, a network simulator that provides realistic and detailed models for network links and protocols (FTP, UDP, wireless traffic, *etc*.). NSGrid delivers highly accurate results but the framework lacks portability and scalability, due to the high level of network details.

In contrast to the discussed grid simulation environments (which suffer from scalability problems and long run times), general-purpose simulators usually provide short simulation times and easy extensibility. Furthermore, elaborate documentation and time-aware debugging mechanisms significantly simplify the creation of new models. However, experience has shown that general-purpose simulation environments (as well as grid-specific simulators) are not suitable for the current study as the possibilities for modelling resource, network and job dynamics are quite limited.

In view of the lack of support for dynamic resource availability there is a clear need for a high-performance simulator dealing with these demanding environments. Therefore the DSiDE (Dynamic Scheduling in Distributed Environments) discrete-event grid simulation framework was developed. DSiDE is developed stepwise starting from a simple grid model consisting of a single Scheduler, a single Information Service and multiple Computational and Storage Resources. The initial model will be refined and extended with checkpointing, resource reservation, job dependencies and network-aware concepts. In the next section the design of DSiDE is discussed in more detail.

DESIGN OF THE DSIDE SIMULATOR

The main requirements for the development of the DSiDE simulator were portability, easy extensibility, short simulation times and flexibility with respect to the implementation of dynamic behavior of grid resources and jobs. The resulting simulator is "flexible" since it can be extended in several ways and can be easily configured without code recompilation. It is also "portable" since it is available for Win32 and Linux, and potentially also other platforms.

DSiDE was developed in C++ because this object-oriented programming language offers a good compromise between efficiency and advanced features such as high-level data constructs, abstract interfaces, smart pointers, exception handling, namespaces, *etc*. Furthermore, the software avoids excessive use of commercial third-party components. In fact, the only such component is Elcel Technology's OpenTop (http://www.elcel.com), which is used for XML parsing. In general, the design of DSiDE is clean, based on the three-tier principle and modern software design patterns (Gamma et al. 2003).

Currently the simulator consists of two separate modules: DExec and DGen (Figure 1). DExec implements a grid simulation environment, containing models for the following grid elements: User Interface (UI), Scheduler, Information Service (IS), Computational and Storage Resources (CR and SR). The UI is responsible for the dynamic generation of jobs with particular characteristics, which are submitted to the Scheduler with a specified frequency. Grid models can contain an unlimited number of UIs. In contrast to the UI, a grid can define only one Scheduler that has as a goal the distribution of jobs among the available resources. Currently, several scheduling algorithms are implemented in DSiDE as dynamically loadable objects: FCFS (First Come First Served), FJFR (First Job First Resource), FJBR (First Job Best Resource), MinMax (submits the smallest job to the fastest resource) and MaxMax (submits the longest job to the fastest resource). Scheduling decisions are based on information collected in each scheduling cycle from an IS. At this moment only one IS can be defined, which can function in two modes: either getting the changed system status immediately, or with a certain

delay due to underlying network and slow information propagation. CRs and SRs execute/store workload received from the Scheduler. Each time a new job is received by a resource, its properties (computational speed, storage space, *etc*.) are modified to be later collected by an IS.

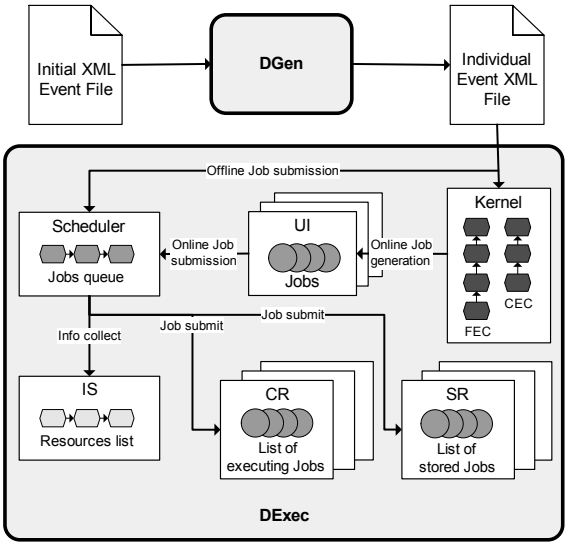

Figure 1: DSiDE Conceptual Diagram

In DSiDE, XML is used for the specification of simulation scenarios. Grid model and workload parameters described in XML serve as an input to the DGen module of DSiDE. All the model modifications are defined as individual Events. Currently the following types of Events are supported:

- **SchedulerAdd**: sets Scheduler parameters (scheduling algorithm, network location, *etc*.).
- **ISAdd**: sets IS parameters (ping interval for retrieving data updates, *etc*.).
- **JobAdd / ResourceAdd**: submits a group of jobs / resources to a grid. Each job / resource belonging to the same group will have similar characteristics. In this way each additional Event can stand for completely different types of jobs / resources, which conforms to the heterogeneous nature of the grid, with its diverse users and resource providers. Attributes of jobs and resources (number, arrival time into the system, execution time, speed, size, number of inputs and outputs, *etc*.) can either be specified as constant values or as a particular distribution which can be chosen from the wide range of supported distributions.
- **ResourceStatusChange**: is responsible for the modelling of the dynamic behavior of grid resources. Resource failure, restart, capacity changes can also occur at fixed timestamps or at times sampled from a specific distribution.

The initial XML Events specification defines groups of events and the way in which they should be generated (offline or at run-time). DGen translates this initial specification into individual Events that will be loaded into the Kernel component of DExec. The Kernel is the heart of the simulator. It consists of two Event chains: FEC (Future Event Chain) and CEC (Current Event Chain). All static and dynamic changes in the simulator status arrive in the form of Events into the FEC where they are sorted by timestamp. During each iteration, the first Event in the FEC is moved to the CEC together with all other Events with the same timestamp. The simulator keeps running until either the End Event is executed, or the simulation end-time is reached.

SIMULATION RESULTS

All simulations discussed in this and the following section were run on the UGent Grid infrastructure (http://begrid.atlantis.ugent.be), consisting of 41 HP DL145 Dual Opteron nodes with 4 GB RAM and 40GB HD space each. All nodes are running Scientific Linux 3.0.5 and version 2.7.0 of the LCG grid middleware (http://lcg.web.cern.ch/lcg). Further, only coarse-grained gridification of DSiDE experiments was performed (*i.e.* a simulation was considered as an undividable unit of work). Fine-grained gridification would entail splitting up a simulation into constituents, which is a complex task that is believed to cause severe overhead. The time to complete each simulation experiment varied from 15 minutes to 3 hours.

A desktop grid model consisting of 10 CRs was used for all simulations presented in this section. Each resource was assumed to process one job at a time and had a constant speed of 1 MIPS (Million Instructions Per Second). Initially, it was decided to model the underlying network with fixed bandwidth and latency.

The concept of a desktop grid is that PCs of individual users are utilized for job execution during idle periods (during a lunch break, meeting, at night). However, when the owner of the machine starts his/her own applications, all external jobs have to be terminated and eventually migrated to another location. This dynamic resource behavior was simulated for a varying frequency and time span of resource unavailability periods. Measurements were performed for a total grid unavailability fluctuating from 0% up to 97%. This implies that desktop resources with high utilization (where only 3% of the time can be spent on external job execution) were considered, next to systems fully dedicated to the execution of the latter. A constant arrival stream of workload was modeled with job submission times uniformly distributed from 1 to 5 minutes. Furthermore, 3 classes of jobs where used: short (10 minutes), medium (1.5 hours) and long (3.5 hours). The grid behavior was observed during 5 weeks of simulated time.

Figure 2 summarizes the outcomes of the performed experiments. The figure depicts the ratio of useful work performed by the grid versus the work lost due to abrupt resource unavailability. It is important to notice that these outcomes are strongly dependent on the way a certain unavailability percentage was acquired. More specifically, the inaccessibility ratio of a resource

depends on two parameters: frequency of "failure" and the time it takes the resource to "restore". Thus, the same result can be achieved in different ways: a resource can "fail" frequently and "restore" fast; or it can "fail" less frequently and restore "slower". Throughout all the simulations described in this paper, values for both parameters were distributed uniformly in the intervals shown in Table 1.

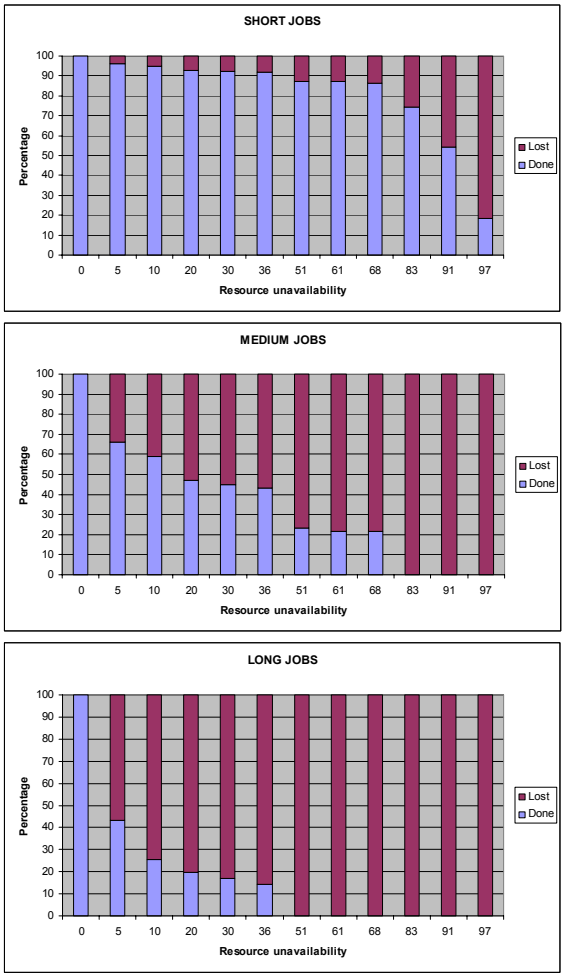

Figure 2: Simulated Grid Performance Results for Different Job Classes

From the results in Figure 2, it is clear that in a heavily loaded system, the longest jobs suffer the most from system instability. When resource "failures" occur rarely (5% of total system time), only 3% of the work will be lost in case of short jobs, 34% in case of medium jobs and 57% in case of long jobs. Further, the frequency of a resource "failure" has a larger effect on grid performance than the time it takes a resource to restore. This can be seen from fast changing ratios between useful and lost work, which occur at points where failure frequency changes. More specifically, in case of medium jobs there is a sudden decrease in the percentage of useful work done when the move is made from a system that is unavailable 36% of its total run time to a grid that is unavailable half of its total run time. At the same time the move from 51% to 61% failure time seems to be less important. This can be explained by the fact that in the first case the failure frequency changes, while the restart frequency remains the same, and in the second case vice versa. The "jump" is larger when the job size increases.

Table 1: Simulated Failure and Restart Frequencies of Grid Resources

Failure frequency	Restart frequency
0	0
0 - 500	0 - 40
0 - 400	0 - 60
0 - 300	0 - 100
0 - 300	0 - 150
0 - 300	0 - 200
0 - 200	0 - 200
0 - 200	0 - 280
0 - 200	0 - 350
0 - 100	0 - 350
0 - 50	0 - 350
0 - 20	0 - 350

With this simple simulation scenario, a quantitative comparison of grid performance degradation due to dynamic changing resource availability for different job classes was given. From the collected results can be concluded that a scheduler with dynamic detection of changes in resource status, combined with either job restart or migration mechanisms, can significantly improve system performance.

CASE STUDY: TORNADO APPLICATION

The aim of the current research is development of efficient application-aware dynamic scheduling algorithms. As a use-case for this research, the Tornado kernel (Claeys et al. 2006a) was chosen. Tornado is an advanced software system for modelling and virtual experimentation with biological systems, which thus far has mainly been applied to water quality processes. Tornado has an object-oriented design and is composed of strictly separated modelling and virtual experimentation environments.

The main elements of the Tornado modelling environment are model compiler and model builder. The model compiler converts models described in a high-level, declarative, object-oriented modelling language to flattened, executable model code. The model builder then compiles and links the executable model code into a binary object that can be dynamically loaded into the experimentation environment (Figure 3).

In the Tornado experimentation environment different types of virtual experiments can be designed, such as simulations, optimizations, sensitivity and scenario analyses, which are run using the executable models developed in the modelling environment. Virtual experiments can be subdivided in two categories: atomic and hierarchically structured (*i.e.* consisting of

one or more sub-experiments), where in the second case the order of execution can be random or fixed.

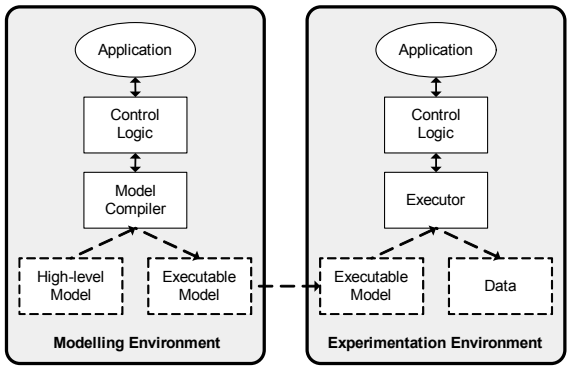

Figure 3: Tornado Conceptual Diagram

Depending on model complexity, simulation intervals and desired accuracy, the execution time of a Tornado experiment varies from several minutes to several days. Therefore it seemed desirable to provide a means for distributed execution of virtual experiments. So far, only coarse-grained gridification was considered, with a simulation experiment as the smallest unit of work.

Currently, Tornado supports semi-automated execution of its jobs in two distributed environments: Typhoon (Claeys et al. 2006b) and LCG-2. Table 2 illustrates the performance improvement gained by an introduction of distributed execution. As an example the scenario analysis experiment for the *Marselisborg* WWTP (Waste Water Treatment Plant) in Denmark was used. The experiment was performed on the UGent Grid infrastructure, described in the previous section.

Table 2: Execution Times for *Marselisborg* on the UGent Grid infrastructure consisting of 41 HP DL145 Dual Opteron Nodes

Software	1 run	1,084 runs
Tornado	6 min	4.5 days
Tornado + LCG-2	6 min	3.5 hours

Distributed execution can significantly speed up the execution of Tornado jobs, however practical experience has shown that there is still a need for more advanced forms of scheduling. From September 2005 until April 2006, measurements of resource availability in the BEgrid (Belgian compute/data grid infrastructure for research) (http://www.begrid.be/begrid.htm) were performed. The result was that the mean availability of the grid's computational resources is about 85% of the measured time, and failures occur approximately every 3 weeks (depending on the resource provider). Since BEgrid and UGent Grid (which is a part of the BEgrid grid) both only support static scheduling of workload, all running jobs are lost, even without owner notification, each time a resource failure occurs. To measure the exact impact of such failures on execution of Tornado experiments, a number of simulations with DSiDE were performed.

The UGent Grid infrastructure is modeled consisting of 41 CRs, each able to run 2 jobs simultaneously. The speed of all resources is set to 2 MIPS and bandwidth of the network is 100 Mbit/sec. Further, the model defines a single Scheduler, running the FCFS scheduling algorithm, and a single IS. This grid model arrangement agrees with the actual architecture of the UGent Grid.

Typical to the behavior of Tornado users is that jobs are submitted once or twice a day in batches of approximately 1,000 jobs. The most common type of Tornado jobs is modeled, where job lengths are normally distributed with an average of 10 minutes and a standard deviation of a couple of minutes. Jobs use inputs of 2 Mbytes and produce 10 Mbytes output data. Table 1 summarizes the failure and restart frequencies that are used to model dynamic resource behavior. The grid is observed during 5 weeks of simulated time.

Figure 4 shows simulation outcomes for 10 Tornado users, generating system workload following the above-described job submission pattern. In case of computational grid resources with about 10% downtime, 7% of all workload will be lost as a consequence of system instability. If the system workload increases, a larger percentage of lost jobs is expected.

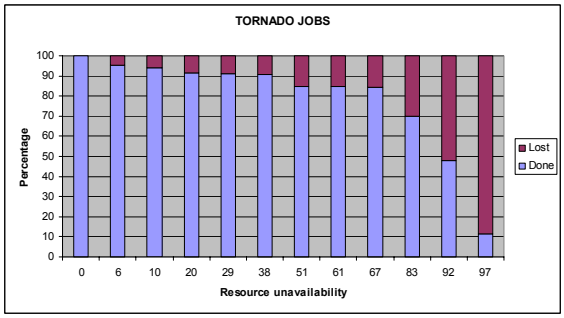

Figure 4: Simulated Grid Performance Results for Tornado Jobs

CONCLUSIONS AND FUTURE WORK

Functional tests and calibration form an essential part of the design of efficient dynamic scheduling algorithms for grids. Most of the time simulation environments are used for this purpose, because evaluating different possible scenarios in real grid systems is overly time-consuming and practically unfeasible. Currently, there exist different discrete-event simulators for grid modelling. The problem with the majority of them is that they do not offer enough flexibility for the representation of dynamics of grid resources and jobs. In reality the latter forms an important feature of grid, with its multiple resource providers and various users. In this paper a design of a new discrete-event simulator for dynamic grids, named DSiDE, was discussed. A number of experiments were performed using this

simulator, in order to show the effect of dynamic resource failure on the execution of various types of jobs.

Resource failure is just one aspect of dynamic grid behavior. Shortly, the effect of other aspects, such as changing resource load, changing job characteristics (length, size of input and output, *etc*.) will be studied. Further, dynamic solutions for the above-stated problems will be proposed.

REFERENCES

Barak, A. and O. La'adan. 1998. "The MOSIX Multicomputer Operating System for High Performance Cluster Computing." *Future Generation Computer Systems*, 13(4-5) (Mar), 361-372.

Buyya, R. and M. Murshed. 2002. "GridSim: A Toolkit for the Modelling and Simulation of Distributed Resource Management and Scheduling for Grid Computing." *Concurrency and Computation: Practice and Experience*, 14(13-15) (Nov-Dec).

Claeys, F.; D.J.W. De Pauw; L. Benedetti; I. Nopens; and P.A. Vanrolleghem. 2006a. "Tornado: A versatile and efficient modelling & virtual experimentation kernel for water quality systems". In *Proceedings of the International Environmental Modelling and Software Conference (iEMSs)* (Burlington, Vermont, Jul 9-13). *Accepted*.

Claeys, F.; M. Chtepen; L. Benedetti; B. Dhoedt; and P.A. Vanrolleghem. 2006b. "Distributed virtual experiments in water quality management." *Water Science and Technology*, 53(1), 297-305.

Foster, I. and C. Kesselman. 1999. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Gamma, E.; R. Helm; R. Johnson; and J. Vlissides. 2003. *Design Patterns.* Addison-Wesley.

Gehring, J. and A. Reinefeld. 1996. "MARS – A Framework for Minimizing the Job Execution Time in a Metacomputing Environment". *Future Generation Computer Systems*, 12(1), 87-99.

Legrand, A.; L. Marchal; and H. Casanova. 2003. "Scheduling Distributed Applications: the SimGrid Simulation Framework". In *Proceedings of the 3st International Symposium on Cluster Computing and the Grid (CCGrid)* (Tokyo, Japan, May 12-15).

Phatanapherom, S.; P. Uthayopas; and V. Kachitvichyanukul. 2003. "Fast Simulation Model for Grid Scheduling Using HYPERSIM". In *Proceedings of the Winter Simulation Conference (WSC)* (New Orleans, Louisiana, Dec 7-10).

Roy A. and M. Livny. 2003. "Condor and Preemptive Resume Scheduling". In *Grid Resource Management: State of the Art and Future Trends*, J. Nabrzyski, J.M. Schopf and J. Weglarz (Eds.). Kluwer Academic Publishers, 135-144.

Thysebaert, P.; B. Volckaert; F. De Turck; B. Dhoedt; and P. Demeester. 2004. "Evaluation of Grid Scheduling Strategies through NSGrid: a Network-Aware Grid Simulator". *Neural, Parallel and Scientific Computations*, 12(3), 353-378.

AUTHOR BIOGRAPHIES

MARIA CHTEPEN was born in St. Petersburg, Russia. She received a MSc in Computer Science from Ghent University in 2004. She is a research assistant with the Fund of the Institute for Innovation and Technology (IWT), at the department of Information Technology at Ghent University. Her main research interest include distributed systems and their performance optimization.

FILIP H.A. CLAEYS was born in Ghent, Belgium. He received a MSc in Computer Science from Ghent University in 1992 and a Master's in Artificial Intelligence from K.U.Leuven in 1995. He currently works as a senior software engineer for HEMMIS N.V. and leads a research group in the field of modelling and simulation software tools at Ghent University.

BART DHOEDT received a PhD degree in Engineering from Ghent University in 1995. In 1997 he became professor at the Faculty of Applied Sciences, Department of Information Technology. His research interests are software engineering and mobile & wireless communications. He is author of approximately 100 peer-reviewed papers.

FILIP DE TURCK obtained his PhD in Electronic Engineering from Ghent University. At the moment, he is a part-time professor and a post-doctoral fellow of the FWO-V, affiliated with the Department of Information Technology of Ghent University. He is author of 80 peer-reviewed papers. His main research interests include scalable software architectures for telecommunication network and service management.

PIET DEMEESTER received the PhD degree from the Ghent University in 1988. In 1992 he started a new research activity on broadband communication networks resulting in the IBCN-group (INTEC Broadband communications network research group). In 1993 he became professor at Ghent University. His research activities cover various communication networks, including network and service management, telecom software, *etc*. He is author of more than 400 publications and a member of the editorial board of several international journals.

PETER A. VANROLLEGHEM, bio-engineer, PhD, is former head of the BIOMATH research team at Ghent University and has ample experience with modeling, monitoring and control of wastewater treatment systems. The development of new sensors, optimal experimental design techniques, and benchmarking of control strategies using simulation have been important scientific contributions, next to modeling and simulation methodologies. He has over 175 peer-reviewed papers and is very active within the International Water Association.