On the use of NSGrid for accurate Grid Schedule Evaluation

Bruno Volckaert, Pieter Thysebaert*, Marc De Leenheer, Filip De Turck†,
Bart Dhoedt, Piet Demeester
Department of Information Technology, Ghent University - IMEC
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium.
Tel.: +32 9 267 35 87, Fax: +32 9 267 35 99
E-mail: Bruno.Volckaert@intec.UGent.be

Abstract *Computational Grids consist of a multitude of geographically distributed resources. The co-allocation of several of those resources allows for the execution of highly computing-intensive and data-intensive jobs. In order to obtain quality schedules (in terms of job response time and resource utilization), different factors such as resource load (computational resource load and bandwidth usage) and data location need to be taken into account. In addition, because of the size of realistic Grids, relevant parameters w.r.t. schedule quality can often only be obtained through simulations. In this paper, we detail how a discrete-event simulator was extended to support the simulation of scheduling both cpu- and data-intensive jobs on a Grid using network-aware scheduling algorithms. These jobs can either pre-stage data, or access data remotely while executing. We examine the case in which resource-to-resource data connections with guaranteed bandwidth can be set up, both when capacitated Virtual Private Networks (for certain job classes) are defined upfront and when connection requests for all jobs are dealt with in a pure First Come First Serve way. Our results show that in order to generate quality grid schedules, it is necessary to take into account network status information in the scheduling algorithms in order to improve the average job response time. Furthermore, we show how making upfront bandwidth reservations induces fair sharing of network resources between different job classes and thus improves the weighted average response time.*

Keywords: Grid Scheduling Strategies, Grid Simulation, Network-Aware Scheduling.

1 Introduction

Computational Grids [1] consist of the aggregation of a dynamically changing set of heterogeneous resources, distributed over several locations. The different resource types involved include computational resources, storage resources, data generating instruments and the interconnecting network. The combined power of the grid's constituent resources can be exploited to handle resource-intensive jobs. Choosing resources on which to execute a job is done by a scheduling algorithm. Due to the typical job sizes (job duration and data sizes) involved, and limited network bandwidth availability between resources (when compared to e.g. a local cluster), network load and data locality are crucial decision factors. Indeed, as input data needs to be present at the execution site at the time it is needed, available bandwidth (or lack thereof) determines the actual starting time of a job when all its input data is pre-staged to its allocated computational resource; if job execution can start in parallel with the transfer of its input data (e.g. that job accesses its input data in a sequential block-per-block way) the job's computational progress rate is also limited by that bandwidth.

Taking a look at the resource-to-resource network connections (the receiving end being the job's execution site) over which job input data is transferred, one can distinguish between several approaches to allocating these connections: they can be set up on demand for all jobs, or alternatively, resources can be connected in a VPN offering per-service bandwidth guarantees. In the latter case, jobs can only request connections within the VPN corresponding to their service type.

For both cpu-intensive and data-intensive jobs, we investigate these different data access and connection setup patterns for different network scenarios. To this end, we have used NSGrid [2], our in-house developed Grid simulator built on top of ns-2 [3]. We have extended it to deal with the concepts described above, and have used it to compare schedules produced by different algorithms (taking into account computational resource load, network

*Research Assistant of the Fund Of Scientific Research - Flanders (F.W.O.-V., Belgium)

†Postdoctoral Fellow of the Fund of Scientific Research - Flanders (F.W.O.-V., Belgium)

bandwidth and job class) in terms of job response time and resource utilization.

This paper is structured as follows: in section 2 we give an overview of related work. Section 3 details the simulation models and components relevant to network-aware scheduling that were introduced in our simulator. Section 4 elaborates on the different Grid scheduling strategies used, while the evaluation of those strategies (using NSGrid) for different job classes in a typical Grid topology is detailed in section 5. Finally, section 6 summarizes the paper and briefly addresses future work

2 Related Work

Grid scheduling is a research area receiving a lot of attention. For instance, distributing work packets for collaborative computing efforts (e.g. SETI [4]) to computational elements is discussed in [5]. Because of the application's particular nature, the grid can be modelled as a tree, with all work packets originating from the root node. Grid scheduling strategies which take both computational resource load and data locality into account are discussed in [6]. The network connecting different sites is not simulated, but it is assumed that the different sites are connected through a VPN-like construction over which TCP communication occurs. Scenarios where files are pre-staged are considered, but data transfers in parallel with job execution are not.

The *Metacomputing Adaptive Runtime System* (MARS [7]) is a framework for utilizing a heterogeneous WAN-connected metacomputer as a distributed computing platform. When scheduling tasks, the MARS system takes into account Computational Resource and Network load, and statistical performance data gathered from previous runs of the tasks. The MARS approach differs from the scheduling model simulated by NSGrid, as NSGrid allows for service-differentiated scheduling.

Existing Grid simulators include Bricks, MicroGrid, SimGrid and GridSim. The Bricks Simulator [8] focuses on client/server interaction in global high performance computing systems. It allows for a single centralized scheduling strategy, which does not scale well to large Grid systems and does not support the notion of multiple (competing) schedulers.

MicroGrid [9] is an emulator modelled after Globus, allowing for the execution of Globus-enabled applications on a virtual Grid system. Research into the area of Grid scheduling algorithms can be cumbersome with this kind of approach, since it requires the construction of an actual application to test.

SimGrid [10] is designed to simulate task scheduling (centralized or distributed) on Grids. Version 1 of SimGrid can be regarded as a toolkit (which interfaces to the C programming language) from which domain-specific simulators can be built. The second version of SimGrid is dubbed *MetaSimGrid* [11] and is essentially a simulator built upon this toolkit to enable the construction of simulations with multiple schedulers (as C programs). Models for network links as well as for TCP connections are present in SimGrid. This validated TCP implementation allows for smaller simulation times when compared to the packet-level TCP simulation performed by network simulators. Of course, simulations using other transport protocols that are not readily available in SimGrid require these protocols to be implemented first.

GridSim [12] is a discrete-event Grid simulator based on JavaSim [13]. This simulator allows simulation of distributed schedulers, and is specifically aimed at simulating market-driven economic resource models. While its computational resource models are highly configurable, only a basic notion of network connectivity is supported and underlying network dynamics are not accurately simulated.

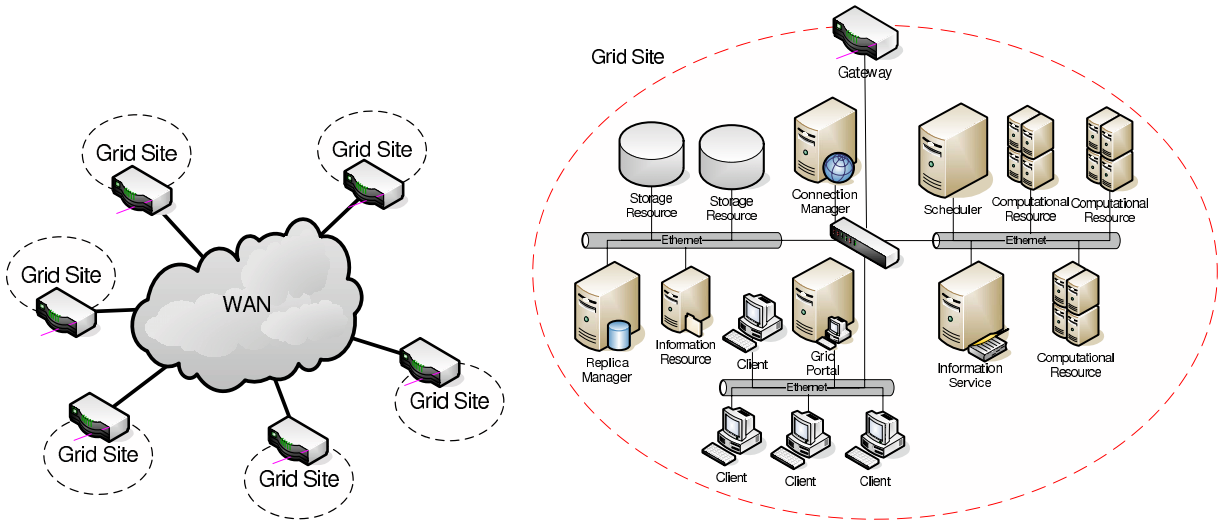
3 NSGrid

The next section gives an overview of the basic Grid components (and their functionalities) as implemented in NSGrid. We focus on the models that are of particular importance for network-aware scheduling; more detailed information on the other models can be found in [2]

3.1 Grid Model

In NSGrid, Grids are modelled as a collection of interconnected and geographically dispersed *Grid sites*. Each Grid Site can contain multiple *resources* of different kinds (their modelling is explained in [2]) such as Computational Resources (CR), Information Resources (IR), Storage Resources (SR) and network resources. A key property of this model is the explicit treatment of the network as a "resource", allowing the scheduler to take decisions based on observed and expected future load of the network interconnecting the different processing/information/storage elements.

At each Grid Site, resource properties and status information are collected in a local *Information Service*. Jobs are submitted through a *Grid Portal* and are scheduled on a collection of resources by a *Scheduler*. To this end, the scheduler makes *reservations* with the *Resource Managers*; in our environment, a *Connection Manager* manages a collection of network links, while the *Computational*, *Information* and *Storage Resources* double as their



(a) Grid Topology

(b) Grid Site decomposition

Figure 1: Grid model

own manager. Each site has a *Replica Manager* that enforces a data replication strategy (caching input data for jobs at a site's local Information Resource). All these entities are attached to their own ns-2 node in the underlying simulated network. LAN links interconnect a Grid Site's local resources, while Grid Sites themselves are interconnected by means of MAN and WAN links.

3.2 Network Model

Interconnections between resources are modelled as a collection of point-to-point connections, each offering a guaranteed total bandwidth available to Grid jobs. Of course, these connections can only be set up if, in the underlying ns-2 network topology, a route (with sufficient bandwidth capacity) exists between the nodes to which these resources are attached. Grid resources can also be interconnected by means of capacitated VPNs: in this case, a VPN tunnel (with guaranteed bandwidth availability) is set up between two Grid resources for a particular Grid job service type. This VPN tunnel carries all connections matching the VPN's endpoints and service type. Such connections can be setup as long as the VPN's residual bandwidth can satisfy the connection demands. VPNs allow for the upfront reservation of bandwidth for a particular (prioritized) service type. The Grid scheduling component we implemented bases its calculations on these connections' bandwidths and makes reservations with resources in such a way that an average throughput for each connection equal to its "guaranteed" bandwidth is obtained. This means that the produced schedules are also correct w.r.t. network bandwidth usage, even if in reality, a network management infrastructure allowing the reservation of end-to-end connections with guaranteed bandwidth is not available.

Connections and VPNs (and the accompanying reservations) in NSGrid are managed by the Connection Manager components. Connection reservations are specified by two distinct endpoints, a service type, maximum delay and guaranteed minimum bandwidth.

Realistic Grid topologies can be manually coded in Tcl, or imported in NSGrid from GridG (a computational Grid generator [14]) output.

3.3 Job Model

The atomic (i.e. which cannot be parallelized) unit of work used throughout this paper is coined with the term *job*. Dependencies between individual jobs (described by a Directed Acyclic Graph) can be expressed using the notion of *Jobgroups*. Each job is characterized by its length (time to execute on a reference processor), its required *input data sets*, its need for *storage* (used for output data), and the *burstiness* with which these data streams are read or written.

Knowing the job's total length and the frequency at which each input (output) stream is read (written), the total execution length of a job can be seen as a concatenation of instruction "blocks". The block of input data to be processed in such an instruction block is to be present before the start of the instruction block; that data is therefore transferred from the input source at the start of the previous instruction block. Similarly, the output data produced by each instruction block is sent out at the beginning of the next instruction block. We

assume these input and output transfers occur in parallel with the execution of an instruction block. Only when input data is not available at the beginning of an instruction block or previous output data has not been completely transferred yet, a job is suspended until the blocking operation completes. A typical job execution cycle (one input stream and one output stream) is shown in figure 2. The presented model allows us to mimic both *streaming* data (high read or write frequency) and *data staging* approaches (read frequency set to 1).

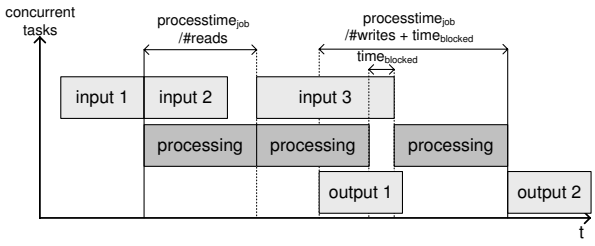


Figure 2: Input-blocking job

3.4 Data Replication

At each Grid Site, a *Replica Manager* can enforce a local replication strategy, meant for lowering overall network usage by replicating data needed by jobs to the site's local Information Resources (IRs): when a job is executing on a CR, and this job gets its input data from a remote IR, the *Replica Manager* will decide whether the input arriving at the CR must be replicated to the local IR. Different replication/replacement strategies can be used for this: least recently used, weighted popularity, etc.

4 Scheduling Algorithms

When jobs are submitted, a *Grid Scheduler* needs to decide where to place the job for execution. The scheduling algorithm used in making this selection has a big impact on Grid performance, and influences overall Grid job throughput, Grid resource efficiency etc.

Our *Grid Scheduler* uses a queuing approach: if the scheduler is unable to allocate the needed resources for a job, the job gets queued for rescheduling in the next scheduling round. Resources are allocated to jobs in FCFS order and are chosen in order to minimize that job's response time. In our implementation, for jobs needing a single input dataset and generating a single output dataset, this best CR-SR-IR combination is found through an exhaustive search. The time between two scheduling rounds can be fixed, but it is also possible to set a threshold (e.g. time limit or number of unscheduled jobs in the queue) which triggers the next scheduling round.

In what follows we will explain the relevant differences between several scheduling strategies when it comes to taking into account network status and job structure information. Once scheduled, our scheduler does not attempt to pre-empt jobs.

4.1 Network Awareness

Network aware scheduling algorithms will not only contact the Information Services (for information about resources that adhere to the job's requirements), but will also query the Connection Manager for information about the status of the network links interconnecting these resources.

The connection manager can inform the Grid Scheduler about the "best" (directed) connections (in terms of bandwidth and delay) that can be set up between two resources for a given service type. Connections are established between a CR and an SR (output data), or between an IR and a CR (input data). When combining CR load information with this network information, the *Grid Scheduler* is able to accurately predict the response time of a job when it is scheduled on some set of resources. If only CR load were taken into account, the accuracy of these predictions would decrease rapidly with higher network loads: as the network becomes a bottleneck, the job will slow down significantly and its response time will increase.

4.2 Resource Locality Preference

When choosing resources, an optimal resource set to be assigned to a job can clearly be found by examining all feasible resource sets. In reality, however, different resources (located at different sites) will not incur equal costs. In absence of a resource cost model, we have taken the following approach : we discriminate between algorithms which look for the overall best resource set (disregarding the exact location of the various resources in the set) and algorithms which prefer resources local to the site at which the job to be scheduled was submitted. Only if these local resources are unable to execute the given job will the latter algorithms start looking for remote (i.e. possibly located at sites different from the local one) resources.

Of course, the exact composition of the optimal resource set will depend upon

- the input data sets required by the job
- the availability of those data sets on the different information resources

Therefore, schedule quality can be increased by use of well-chosen data replication strategies (independent of the job scheduling strategies). We refer to section 3.4 for the approach implemented in NSGrid.

5 Simulation Results

5.1 Simulation Parameters

5.1.1 Topology

A fixed Grid topology was used for all simulations presented here. This topology consists of 12 Grid sites (each having a single CR, SR and IR) interconnected by means of WAN links (bandwidth of these WAN links is parameterized in our simulations). Each site also has its own *Information Service* (storing Resource properties and status) and local *Grid portal* (through which users can submit jobs). Local resources are connected through 1Gbps LAN links.

To reflect the use of different tiers in existing operational Grids, not all Computational Resources are equivalent: the least powerful CR has two processors (which operate at the reference speed). A second class of CRs has four processors, and each processor operates at twice the reference speed. The third - and last - CR type contains 6 processors, each of which operates at three times the reference speed. Conversely, we have 2 top class CRs, 4 second class CRs and 6 low-end CRs (containing reference processors).

Since we have have focused on determining the influence of the use of network resource status on the schedule calculation, we have assumed that SRs offer "unlimited" disk space that can be read and written at "sufficiently high" speed (i.e. higher than the needed data transfer bandwidths).

The use of different dynamic data replication strategies has been studied extensively in [6]. Therefore, we have replicated 6 possible data sets over the IRs in a static way. Each IR contains 3 out of these 6 possible data sets, and 50% of the jobs submitted to a site can have local access to its needed data set.

5.1.2 Job parameters

We have used two different job types in our simulations; one is more data-intensive (i.e. higher data sizes involved), while the other is more cpu-intensive. At each Grid Site, two "clients" have been instantiated, one for each job type. Each client submits mutually independent jobs to its Grid Portal following a Poisson process. All jobs need a single IR and a single SR. The ranges between which the relevant job parameters vary have been summarized in table 1. Both job types make up 50% of the job load; in each simulation, the job load consisted of 1200 jobs.

	CPU Jobs	**Data Jobs**
Input(GB)	0.01-0.02	1-2
Output(GB)	0.01-0.02	1-2
Arrival Rate(1/s)	0.005-0.01	0.005-0.01
Ref. run time(s)	400-1200	200-400

Table 1: Relevant job properties

For each scheduling algorithm, we have chosen to use a fixed interval between scheduling rounds of $100s$. From the arrival rates in table 1 and the fact that multiple sites submit jobs simultaneously, it follows that we are likely to find multiple jobs in the queue at the start of each scheduling round.

5.2 Schedule Evaluation

5.2.1 Average Job Response Time

We define the *response time* of a job as the difference between its end time and the time it is submitted to the scheduler. This metric has been used to compare all schedule instances with.

5.2.2 Input Data Pre-staging

In figure 3, we have plotted the average job response time in four cases, differentiating between

- whether or not data is pre-staged (results for pre-staging show up as the upper curve pair)
- whether or not local resources were preferred (the uppermost curve of each pair)

Clearly, applications structured in such a way that they can make use of the execution/transfer parallelism have better response times. The disadvantage of preferring local resources over the overall best resource set is limited. In fact, if the amount of data-intensive jobs increases (for this simulation, half of the jobs was data-intensive), it can be expected that this disadvantage becomes even smaller, as data-intensive jobs benefit most from being scheduled on geographically close resources (due to replication and higher LAN bandwidth).

5.2.3 Influence of capacitated VPNs

If data connections are setup on demand using a pure FCFS scheme, it is likely that data-intensive jobs will quickly consume all the available bandwidth, causing cpu-intensive jobs to remain queued for a longer period of time.

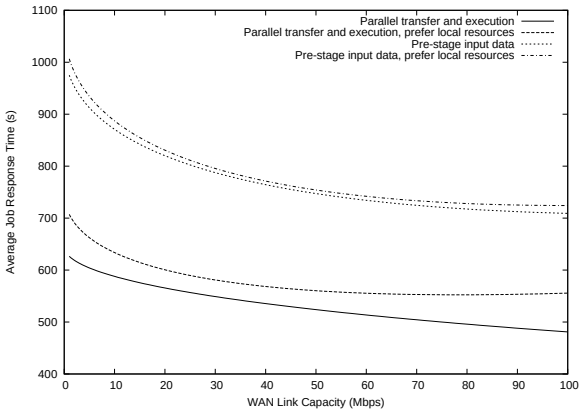


Figure 3: Response Time: influence of pre-staged input

The upfront reservation of bandwidth to each job type ensures that these cpu-intensive jobs will never be excluded from remote execution (i.e. move towards a faster Computational Resource). We have simulated our job load again, but this time we have setup VPNs for the two job classes (data-intensive vs. cpu-intensive). We reserved more bandwidth for the data-intensive jobs ($20\% - 80\%$ ratio). The job response time for the different algorithms in this scenario is shown in figure 4. This approach visibly improves the response time: cpu-intensive jobs do not remain queued for an extraordinary period of time, and as these jobs have high run times, this has a significant impact on the average job response time (response times in case capacitated VPNs have been setup are shown in the bottommost two curves, the higher one pointing to the case where local resources were preferred).

Further fine-tuning of VPN parameters to job characteristics (interarrival time, run time, bandwidth requirements) might further improve the average job response time.

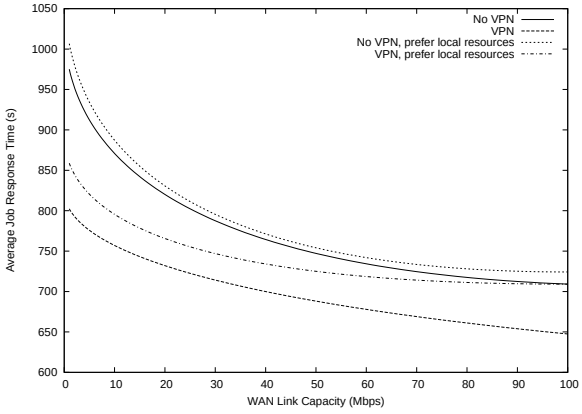


Figure 4: Response Time: influence of capacitated VPNs

6 Conclusions and future work

In this paper we detailed important extensions to a discrete-event Grid simulator to evaluate job schedules taking into account network information. These extensions include the ability to model different connection setup patterns, the setup of capacitated VPNs, and the ability to model jobs which either pre-stage all data or have live access to remote data. Our simulation results show opportunities to generate significantly better schedules. In particular, it follows that upfront reservation of bandwidth (between Grid resources) for certain job types can improve the response time by avoiding that data-intensive jobs monopolize available bandwidth. In the cases studied, a reduction in average job response time of about 10% was typically found.

In the near future, we plan to investigate the quality of schedules produced by the network aware algorithms under different Information Resource data set replication strategies. In addition, these algorithms will be evaluated for more elaborate Grids. From the results presented in subsection 5.2.3, it seems worthwhile to pursue more refined network aware algorithms that can take into account a job's service class, priority and computation/communication ratio. Due to service class differences, we believe that the best results can be obtained by applying algorithms specifically tailored to schedule all jobs in one particular service class.

References

[1] I. Foster, C. Kesselman, "*The Grid: Blueprint for a New Computing Infrastructure*", Morgan Kaufmann (1999)

[2] B. Volckaert, P. Thysebaert, F. De Turck, P. Demeester, B. Dhoedt, "*Evaluation of Grid Scheduling Strategies through a Network-aware Grid Simulator*", Proc. of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (2003), Vol. 1, pp. 31-35

[3] S. McCanne, S. Floyd, "*ns Network Simulator*", http://www.isi.edu/nsnam/ns

[4] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthim, "*SETI@home: An Experiment in Public-Resource Computing*", Communications of the ACM, (2002), Vol. 45 No. 11, pp. 56-61

[5] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert, "*Bandwidth-centric allocation of independent tasks on heterogeneous*

platforms", Technical Report 4210, INRIA, Rhone-Alpes, (2001)

[6] K. Ranganathan, I. Foster, "*Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids*", Journal of Grid Computing, Vol. 1, Issue 1, Kluwer Academic Publishers (2003), pp. 53-62

[7] J. Gehring, A. Reinfeld, "*Mars - a framework for minimizing the job execution time in a metacomputing environment*", Proc. of Future General Computer Systems '96 (1996)

[8] Takefusa, A., Casanova, H., Matsuoka, S., Berman, F., "*A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid*", Proc. of 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), (2001), pp. 406-415

[9] Song, H.J., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., Chien, A., "*The MicroGrid: a Scientific Tool for Modeling Computational Grids*", Proceedings of SC2000, Dallas, Texas, (2000)

[10] Legrand, A., Marchal, L., Casanova, H., "*Scheduling Distributed Applications: the SimGrid Simulation Framework*", Proc. of the 3rd International Symposium on Cluster Computing and the Grid, (2003) pp. 138-145.

[11] Lerouge, J., Legrand, A., "*MetaSimGrid : Towards realistic scheduling simulation of distributed applications*", ENS-LIP Research Report nr. 2002-28, (2002)

[12] Buyya, R., Murshed, M., "*GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*", The Journal of Concurrency and Computation: Practice and Experience (CCPE), Wiley Press, (2002)

[13] Tyan, H. Y., Ju Hou, C. J., "*JavaSim: A component-based compositional network simulation environment*", Communication Networks And Distributed Systems Modeling And Simulation, (2001)

[14] Lu, D., Dinda, P., "*GridG: Generating Realistic Computational Grids*", Performance Evaluation Review, (2003), Volume 30, Number 4