Scientific Programming 19 (2011) 231–251
DOI 10.3233/SPR-2011-0332
IOS Press

GSSIM – A tool for distributed computing experiments

Sławomir Bąk [a], Marcin Krystek [a], Krzysztof Kurowski [a], Ariel Oleksiak [a,*], Wojciech Piątek [a] and Jan Węglarz [a,b]

[a] *Poznań Supercomputing and Networking Center, Poznań, Poland*
[b] *Institute of Computing Science, Poznań University of Technology, Poznań, Poland*

Abstract. In this paper we present the Grid Scheduling Simulator (GSSIM), a comprehensive and advanced simulation tool for distributed computing problems. Based on a classification of simulator features proposed in the paper, we define problems that can be simulated using GSSIM and compare it to other simulation tools. We focus on an extension of our previous works including advanced workload generation methods, simulation of a network with advance reservation features, handling specific application performance models and energy efficiency modeling. Some important features of GSSIM are demonstrated by three diverse experiments conducted with the use of the tool. We also present an advanced web tool for the remote management and execution of simulation experiments, which makes GSSIM the comprehensive distributed computing simulator available on the Web.

Keywords: Distributed computing, simulations, resource management, scheduling

1. Introduction

Scheduling algorithms in distributed computing systems have been the subject of intensive research over the last decade. However, research experiments evaluating and making comparative analysis of these algorithms are often difficult to be conducted. This is caused by many problems including, for example, difficulties in obtaining exclusive access to large-scale infrastructures for research purposes or the lack of certain functionalities of real resource management systems, such as advance reservation (AR). Furthermore, emergence of new computing paradigms such as grids, clouds, multi-core processing, etc. caused gaining importance of various aspects of distributed computing, for example virtualization and energy efficiency issues. Therefore, due to diversity of distributed resource management systems and a significant technical effort needed to establish large-scale computing infrastructures, simulations are commonly used to evaluate candidate algorithms and architectures. However, most of simulations have been developed for a specific purpose, so that they cannot be re-used elsewhere. For this reason, several generic simulation tools were introduced such as SimGrid [6], GridSim [4], etc. Some of these tools provide a good basis for implementation and simulation of a wide range of algorithms. Nevertheless, in most cases developers must implement experiments by themselves using just the basic functionality of simulators. In consequence, setting up an experiment also requires a lot of work and is rarely applicative by other researchers.

To address these issues, we introduced the Grid Scheduling Simulator (GSSIM) which provides an automated framework for the management of experiments related to resource management in distributed computing environments [17]. GSSIM achieves it through a flexible design of architecture and interactions between scheduling components, a possibility of plugging scheduling algorithms into the simulated environment, modeling synthetic workloads and adopting real traces in popular formats, a configuration of the computing infrastructure topology both on logical and physical level, and many other features as further described in detail.

The GSSIM framework is complemented by the portal which enables online access to the simulator via a user-friendly experiment editor, workload generator and experiments repository. The rich web interface allows executing the simulation experiments remotely, provides access to workloads, resource descriptions and implementations of algorithms, and enables in-

*Corresponding author: Ariel Oleksiak, Poznań Supercomputing and Networking Center, ul. Noskowskiego 10, 61-704 Poznań, Poland. E-mail: ariel@man.poznan.pl.

teractive visualization of results. In this way, GSSIM provides a comprehensive environment enabling researchers to test resource management algorithms and architectures, and to exchange not only workloads but also results of experiments and implementations of algorithms.

The remaining part of the paper is organized as follows. In Section 2 we propose a classification of simulator features and, on this basis, we present GSSIM in comparison with other simulation tools. Section 3 provides the details of the specific features of GSSIM. In particular, it discusses the simulated architecture, introduces the workload management concept, explains how to incorporate a specific application performance model into simulations, describes an extension for flow-based simulation of network with advance reservations, presents energy-efficiency modeling and demonstrates an advanced web interface for remote management of experiments. Section 4 presents examples of experiments that illustrate collecting, analysis, and visualization of results. Final conclusions are given in Section 5.

2. Simulation of distributed computing systems

A wide spectrum of distributed computing problems may need simulations to analyze their properties and possible solutions. Proper modeling of these problems requires a number of features that must be provided by simulation tools. In order to compare simulators with respect to their features and supported distributed computing problems, we summarize the popular scheduling problems common in distributed computing systems in Section 2.1 as well as introduce a classification of simulation features in Section 2.2. On this basis, available simulators are compared in Section 2.3 while summary of problems that can be simulated with the use of GSSIM is presented in Section 2.4.

2.1. Scheduling problems

There is a large number of scheduling problems studied in the literature for many years. Attempts to classify and analyze them can be found in [2,31]. The most important classes have been distinguished and listed below.

On-line and off-line scheduling. In the former case, at a certain point in time there is no information about future jobs. The latter case assumes knowledge about all jobs during scheduling.

Clairvoyant and non-clairvoyant scheduling. In most of real situations exact job execution time is unknown a priori. However, sometimes scheduling is based on execution times given by users or estimated on the basis of previous runs (a lot of algorithms in scheduling theory assume knowledge of execution times). These cases are referred as non-clairvoyant and clairvoyant scheduling, respectively.

Time constraints. If job processing times are known (or estimated), requirements concerning time, or time constraints, can be used. The most common time constraints in literature include ready times (the earliest job start time), due dates (preferable job completion time) and deadlines (the latest job completion time).

Workload types. Depending on a distributed computing infrastructure and scheduling algorithms to be studied researchers may require diverse structures of workloads. Classification of workloads and their main properties are proposed in Fig. 1 and summarized below.

- *Parallelism*. Single tasks may be sequential, parallel (running on multiple processors), distributed (running on multiple nodes) and distributed cross-domain (can run on machines in multiple administrative domains, or sites).
- *Number of processors*. According to the classification given in [8] tasks can be also divided into rigid, moldable, and malleable ones. First two types of task use a constant number of CPUs either specified exactly in advance (rigid tasks) or set at the beginning of task execution (moldable tasks). Number of exploited processors by malleable task may change during the runtime.
- *Preemption*. Tasks with preemption possibility can be suspended during their execution and resumed later on (possibly on another node). Non-preemptive tasks must run until they finish. Otherwise they must be restarted.
- *Time dependencies*. Tasks may contain time dependencies to other tasks. If they have no dependencies they are called independent tasks. Otherwise, they are tasks with preceding constraints, or simply workflows. In the literature various structures of workflows can be found such as chains, trees, uniconnected activity networks, and general (arbitrary).
- *Number of tasks*. Workloads may contain single tasks submitted by specific users or allow submitting multiple tasks by a single user, e.g., the so-called parameter sweep jobs or workflows.

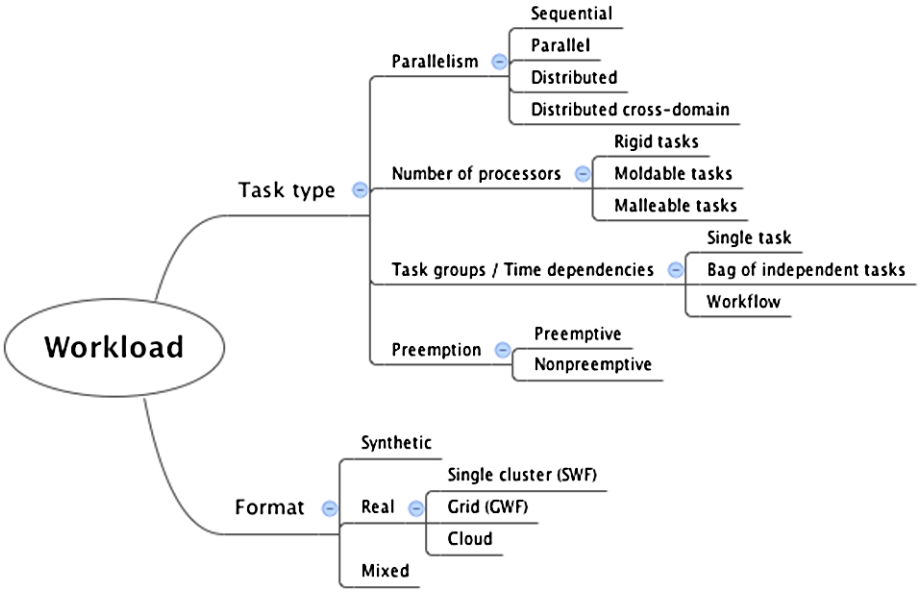

Fig. 1. Properties of workloads. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0332.)

Quality of service. One of important aspects of some distributed systems such as Grids is lack of full control over all resources. For this reason, one must take into consideration the impact of local policies and other users on results of scheduling and either manage jobs in queues or use advance reservation techniques to deliver requested quality of service (QoS). In the first case, start time is not guaranteed a priori and jobs usually are waiting in queues. In the second case, resources are leased, using advance reservation, for a certain period and users obtain information about a start time of the reservation.

Space-shared and time-shared scheduling. Jobs can be assigned to specific multiple processors and machines or scheduled and rescheduled in time. The former is called space- while the latter time-shared scheduling.

Adaptive scheduling. Scheduling problems can be also classified in terms of their adaptation to changing conditions. Sorting from the least to the most adaptive case there are algorithms which do not assume rescheduling at all, assume rescheduling before runtime (when jobs are pending in a queue or an advance reservation for them is established), and runtime rescheduling including suspending, resuming and migrating jobs [19].

Heterogeneity of resources. In addition to properties of jobs and scheduling systems themselves, various types of resources must be considered to provide a simulator applicable for large scope of scheduling problems. First of all, resources may be homogeneous or heterogeneous. Homogeneous resources can be modeled as identical processors, i.e. processors of the same speed. Otherwise, for heterogeneous resources, the execution time is usually set as proportional to different fixed speed of particular processors (uniform processors). However, more complex dependencies between resource properties and execution time are possible, for instance, processing times of tasks on different processors can be arbitrary (unrelated processors).

Evaluation criteria. Significant number of criteria can be applied to evaluate schedules. Among them there are arbitrary global criteria such as makespan, mean flow time, resource utilization, etc., and user-specific criteria that can be defined for individual users. Additionally, criteria related to cost, energy consumption, reliability, etc., are gaining recently importance.

2.2. *Simulation features*

Due to the complexity and costs of building and operating testbeds, extensive research has been conducted in the area of computer-based simulation tools. As a result, a wide variety of simulation tools emerged. A comprehensive taxonomy for design of simulation tools to model large and distributed systems has been presented in [29]. However, to compare various simulation tools more detailed classification of simulation features is needed. In this section we propose a classification that includes a variety of aspects that should be

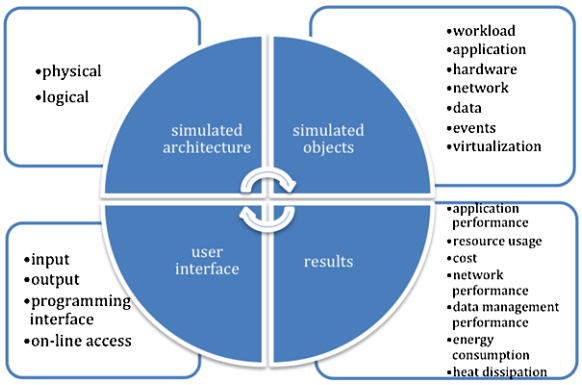

Fig. 2. Essential features of distributed computing systems simulations. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0332.)

taken into consideration in a simulation of modern distributed computing systems. Most of these properties are essential to simulate the aforementioned scheduling problems. The summary of features of distributed computing simulators is illustrated in Fig. 2 and explained in more detail below.

- *Simulated architecture*. Simulation tools may differ in terms of architectures of modeled systems as well as their levels of details. The *logical architecture* of distributed systems may assume centralized control of resources, hierarchical topology of schedulers or fully distributed models. In addition, the *physical topology* may span from computing nodes through racks and clusters up to whole data centers or geographically distributed grids and clouds. Granularity of such topologies may also differ from coarse-grained to very fine-grained modeling single cores, memory hierarchies and other hardware details. Ideally, simulation tools should provide the user with enough flexibility in defining system architecture and hierarchy of resources (including both physical and logical structure).
- *Simulated objects*. Simulation environment ought to enable, apart from simulation of resource components, modeling the behavior of other distributed computing entities. Simulations may include models of various distributed computing entities. One of the most important element of simulation is workload which may be taken from real systems or generated synthetically. *Workload* may contain jobs ranging from single sequential jobs, through parallel and distributed jobs, up to the whole workflows containing time dependencies between jobs. Workload types and elements are summarized in Fig. 1 and discussed in previous subsection. More detailed simulations of jobs may require to model *application* performance by taking into account a lot of factors that affect application execution, e.g., processing unit architecture, application characteristics, and input data. Besides complex information about simulated *hardware* (concerning architecture, characteristics, state, energy profile) and *network* (with different network models), a simulation tool may allow performing specific actions on resources, for instance change their states or support advance reservation (AR). It is also important to permit experimental studies that involves management of large amounts of *distributed data* as well as to model the dynamic nature of distributed environments by handling different *resource events*. Modern simulation tools may also support aspects that gained special attention recently, in particular *virtualization* and energy-aware techniques.
- *Results of the simulation*. The goal of each simulation run is to provide a set of results that allow to evaluate a specific distributed computing system or scheduling method. These results may include: *application performance metrics* (e.g., execution time, flow time, waiting time, completion time), *resource usage*, *cost*, *network performance*, *data management performance*, *energy consumption* and *thermal effects* caused by computations.
- *User interface*. The user interface determines how the user interacts with the simulation tool. The interface is essential to achieve high experiment execution automation and ease of use. First of all, *programming interface* may differ between simulators that may be delivered along with programming libraries or more structured frameworks. Usually frameworks allow to reduce the complexity of the simulation environment and to enter any modifications more conveniently. In addition to programming interface, simulators may be accompanied with intuitive *graphical user interface* that allows to define simulation parameters much faster and easier than using ordinary configuration files. These graphical interfaces may include user friendly representation of simulation results to facilitate the analysis of the distributed computing system. Simulation tool may be also complemented by the *Web interface* in order to allow remote experiments execution or/and provide repository facilities.

2.3. Comparison of simulation tools

Tables 1 and 2 present the results of a comparative analysis of simulation tools based on the classification of simulator features proposed in Section 2.2. Simulation tools were selected to analysis based on their publications, availability, and list of features. Contents of the tables are based on the recent publications concerning the given tools. Moreover, all features were verified by experimental studies and a code analysis of the available simulators (only DGSim was not available to download at the time this survey was compiled).

GridSim [4] developed at the University of Melbourne, is a toolkit that provides means for modeling and simulation of base components that constitute parallel and distributed computing environments (grid users, applications, resources, schedulers and resource brokers) and for the study of the involved scheduling algorithms. A flexible and extensible architecture allows to modify components behavior or incorporate new ones into the existing infrastructure. Thus, GridSim is commonly adopted by other simulation tools that benefit from its core functionality.

Alea [13] developed at Masaryk University in Brno, is the Grid and cluster scheduling simulator. Alea is based on the GridSim toolkit and extends the original basic functionality by introducing some innovative solutions like, e.g., "on the fly" job reading that leads to a better scalability. Simulator is able to cope with general resource management problems like the heterogeneity of jobs, resources, and the dynamic runtime changes. It also supports also a specific workload format – MetaCentrum [22].

MaGate [11] simulator is a simulation-based implementation for the MaGate scheduler. Its goal is to provide a set of easy-to-use decentralized grid schedulers (which are able to collaborate with external grid services) and help researchers to study and evaluate different scheduling algorithms/models and workflows within various scenarios. Simulation process is supported by a grid network overlay simulator that provides services such as group communication and resource discovery.

SimGrid [6] is a joint project between the University of Hawaii at Manoa, LIG Laboratory in Grenoble and University of Nancy. It aims to provide core functionalities and facilities for the simulation of parallel and distributed applications in heterogeneous distributed environments. In particular, SimGrid provides programming environments to support both researchers who study their algorithms and need to run simulations quickly as well as developers who can develop a real distributed applications.

OptorSim [1,5] was initiated as part of the European DataGrid project. It is a modular simulation framework that enables users to perform experimental studies of optimisation strategies under different Grid configurations. In particular, OptorSim allows the analysis of various data replication algorithms and their impact on the resource usage and job throughput in HEP data grids.

DGSim [12] development led by the Delft University of Technology, aims at system and workload modeling in grid resource management architectures. DGSim focuses on automating experiment setup, management and optimizing the overall simulation process. Moreover, it introduces the concepts of job selection policy and grid evolution, which models static changes in the Grid infrastructure considered in the long-term perspective. DGSim provides also some innovative solutions concerning grid inter-operation, grid dynamics and workload modeling.

GroudSim [25] toolkit is developed at University of Innsbruck. It allows simulating both Grid and Cloud computing. The main GroudSim features include: simulation of file transfers, the calculation of costs and background loading. Event module supports resource and network failures as well as recovery events affecting these entities. Simulations can be easily extendable by any kind of probability distribution package.

Conclusions. In this section we review several approaches to simulation of distributed computing systems based on the proposed classification of simulator features. As shown in Table 2, most simulators basically focus on addressing specific problem areas. Since their general goals are different, they vary in terms of simulated architectures and usually allow modeling and simulating only subset of distributed computing entities. Although almost all of the aforementioned toolkits are able to provide complete information about simulated environment during the simulation, there is a lack of tools that allow performing specific actions on resources and in this way affecting their behavior. Moreover, there is a very little support for application performance modeling and incorporation of virtualization techniques. Most of simulation tools handle different resource events, mainly concerning resource dynamics or resource failures. One should note that among simulators discussed in this paper, simulation of distributed data management is closely related to the network modeling capability and thus, these features are either both supported or both not. Several tools

Table 1
Comparison of simulation tools according to the simulated objects

Tool/property	Workload		Application	Hardware	Network	Data	Events	Virtualization
	Input	Model						
GSSIM	Real (SWF, GWF) + synthetic (generator)	Sequential, parallel, distributed, rigid, moldable, preemptive jobs, workflows	Performance modeling	Information (architecture, characteristics, state, energy profile) + control (energy management, state) + AR	Flow model + AR + background traffic generator	File transfer	Resource dynamics + resource (FTA) and network failures, recovery + security	Yes
GridSim	Real (SWF, user-defined – primitive)	Sequential, parallel, distributed, rigid, moldable, preemptive jobs	No	Information (architecture, characteristics, state) + AR	Flow, packet model + background traffic generator	Data storage + file transfer	Resource failures (FTA)	Yes (CloudSim – separate tool)
Alea	Real (SWF, GWF, Metacentrum)	Sequential, parallel, rigid, moldable jobs	No	Information (architecture, characteristics, state)	No	No	Resource failures	No
MaGate	Real (GWF) + synthetic (primitive) generator)	Sequential, parallel, rigid, moldable job	No	Information (architecture, characteristics)	No	No	Resource dynamics	No
SimGrid	XML format (own)	Sequential, parallel, distributed jobs, workflows	Performance modeling	Information (architecture, characteristics, state)	Flow, packet model + background traffic generator	File transfer	Resource dynamics + resource failures	No
OptorSim	Text format (own)	Sequential jobs	No	Information (architecture, characteristics, state)	Flow model + background traffic generator	Data storage + file transfer	No	No
DGSim	Real (SWF, GWF) + synthetic (generator)	Sequential, parallel, distributed, rigid jobs, workflows	No	Information (architecture, characteristics, state)	No	No	Grid dynamics + evolution model	No
GroudSim	Real (GWF)	Sequential jobs	No	Information (architecture, characteristics, state)	Flow model	File transfer	Resource and network failures, recovery	Yes

Table 2
Comparison of simulation tools according to the simulated architecture, user interface and results of the simulation

Tool/property	Simulated architecture		User interface				Results
	Physical	Logical	Programming interface	Input	Output	On-line access	
GSSIM	User-defined	Centralized, decentralized, hierarchical	Framework (introduced layers, plugins)	Comprehensive GUI – experiment editor	Text statistics, interactive charts	Remote experiment management, workload, scheduling algorithms, experiments repository	Application performance, resource usage, network performance, energy consumption, heat dissipation
GridSim	Grid/cluster/computing node/cpu	Centralized, hierarchical	Generic library	Configuration files	Text statistics, charts	No	Application performance, cost, resource usage
Alea	Grid/cluster/computing node/cpu	Centralized, hierarchical	Framework (introduced layers, plugins)	Configuration files	Text statistics, charts	No	Application performance, resource usage
MaGate	Grid/cluster/computing node/cpu	Decentralized	Framework (plugins)	Configuration files + command line or GUI	Text statistics, charts	No	Application performance, resource usage
SimGrid	Cluster/computing node/cpu	Centralized, decentralized	Generic library	Configuration files generator	Text statistics, charts	No	Application performance, cost, resource usage, network performance
OptorSim	Grid/cluster/cpu	Centralized, hierarchical	Framework	Configuration files + command line or GUI	Text statistics, charts	No	Application performance, resource usage, data management performance
DGSim	Grid/cluster/cpu	Centralized, decentralized, hierarchical	Framework	Configuration files	Text statistics, charts	Workload repository	Application performance, resource usage
GroudSim	Grid/cluster/node	Centralized	Framework	Configuration files	Text statistics, charts	No	Application performance, cost, resource usage

accept workloads in various formats and are capable of supporting more sophisticated models than simple sequential jobs. Simulators are also evolving towards user-friendly tools by offering means to visualize the simulation results. Nevertheless, some of them still require to be manually provided with input data as plain-text configuration files.

Compared to other tools, GSSIM allows simulating a wide scope of physical and logical architectural patterns. In particular, GSSIM provides means of simulating complex distributed architectures containing models of the whole data centers, containers, racks, nodes, etc. While many of the presented tools focus on addressing specific issues, GSSIM supports simulations of a wide variety of entities. It covers most of objects simulated by other tools and adds some innovative features such as network advance reservation or generation of events related to resources, network, and security events, which are unique among all simulators. It also provides means (by definition of specific plugins) for complex modeling of the expected application performance. As far as programming interface is concerned, most of popular available tools (e.g., GridSim and SimGrid) are generic libraries supporting only the core functionalities. They are flexible but require substantial effort to develop experiments. GSSIM provides an easy-to-use framework to facilitate this process by several types of plugins, grouping experiments, and workload management. Moreover, GSSIM provides an advanced graphical user interface containing intuitive editors for input data modeling and comprehensive statistics presented in a user-friendly and interactive manner. GSSIM is the only distributed computing simulator enabling remote experiment execution on the Web. GSSIM offers the most comprehensive simulation environment that should satisfy not only grid researchers but also cloud, data center and network administrators or application users/developers. In the following subsection we summarize scenarios and scheduling problems addressed by GSSIM, while the above features, being the main advantages of GSSIM, are described in Section 3 in more detail.

2.4. Scenarios and scheduling problems in GSSIM

GSSIM can be used to simulate the vast majority of scheduling problems in distributed computing systems. It enables simulations of many scheduling strategies applied to various types of applications. In particular, it can simulate scheduling of multiple independent jobs at once, various kinds of parallel jobs, and whole workflows. Moreover, GSSIM is able to handle rigid and moldable jobs, as well as preemptive jobs. Through the appropriate generation of workload (with job arrival times) and implementation of scheduling plugins, an analysis of both on-line and off-line strategies is possible. In addition to types of problems related to workload properties, GSSIM allows defining time constraints in workloads and taking them into account in scheduling algorithms. In clairvoyant scheduling the knowledge concerning execution times and possible constraints has a significant impact on chosen scheduling algorithms and usually requires scheduling with advance reservation in distributed computing infrastructures. As for advance reservation, GSSIM enables simulating of various scheduling problems including both best-effort and QoS-based approaches. To realize the latter case, GSSIM supports negotiations between global schedulers and resource providers that are built on top of local schedulers. This advance reservation mechanism includes two phase commit protocol. More details concerning advance reservation and negotiations in GSSIM can be found in [17,24]. Additionally, GSSIM provides the possibility of scheduling based on performance estimations. These estimations can be generated on the basis of processing times included in the workload or using a custom algorithm implemented by a researcher. Furthermore, more complex dependencies between resource parameters and execution time can be modeled by the implementation of time estimation plugins. In this way GSSIM also deals with a heterogeneity of resources. The details of execution time estimation are presented in Section 3.3. At a local level, a developer of a scheduling plugin has an unlimited access to queues and running tasks. Therefore, a variety of both space- and time-sharing policies can be applied. Moreover, a wide range of adaptive scheduling paradigms is supported, including suspending, resuming, and migrating jobs both before and during the execution. Hence, for instance, developers can implement algorithms on the basis of backfilling and preemption. For each experiment, detailed results are collected. They contain many basic metrics commonly used when evaluating scheduling algorithms, e.g., makespan, mean job completion time, max tardiness, resource utilization, etc. In addition to global criteria, GSSIM supports user-specific criteria which can be defined in the workload for each user separately. In this way, a researcher can study strategies aiming at finding schedules that takes perspectives of individual users into account.

GSSIM has been successfully applied in a substantial number of research projects and academic stud-

ies. For instance, GSSIM allows Grid computing researchers to study the performance of many scheduling algorithms in complex distributed computing infrastructures. In particular, within GSSIM it is possible to establish a wide variety of distributed computing architectures and perform repeatable experiments involving diverse scheduling strategies within GSSIM. They may vary from simple job scheduling policies, through multi-criteria resource management [21], up to complex scheduling problems with QoS requirements as presented in [24] and Section 4.3. GSSIM can also facilitate the work of queuing systems administrators by providing means of evaluating efficiency of various queue configurations. Different approaches to this problem, containing studies on various queues types, increasing number of queues, and various job selection policies, can be found in [18] and Section 4.1. As a network simulator, GSSIM has been used to study the problem of resource allocation with network resources for workflow applications that is presented in [23]. It has also been adopted as a simulation framework in European project called Federica [7] in order to test different approaches concerning resource allocation in virtual network architectures. Moreover, it can also be exploited by network administrators to find possible bottlenecks in network configuration used for demanding HPC applications. Data center administrators or owners can benefit from GSSIM energy module as it provides a comprehensive support for energy-aware scheduling experiments. In this way they can investigate how energy consumption depends on workload type and scheduling policies. Detailed information concerning energy module is presented in Section 3.5. Recently developed extensions provide new interesting features for both cloud providers and cloud users. The former are able to optimize their cloud environment by tuning management policies and configuring virtual machines, while the latter can estimate the cost s of leasing resources from cloud providers. Finally, distributed applications developers have the possibility of tuning their applications to specific computing infrastructure.

3. Simulator features

The comparison of simulation tools presented in Section 2 shown that GSSIM enables modeling and simulation of a wide spectrum of distributed computing problems. GSSIM addresses this issues with the set of sophisticated features described in previous section. Due to modular framework architecture, that corresponds to the real world, a wide scope of modules that provide aforementioned capabilities can be incorporate into GSSIM environment. Details concerning GSSIM architecture, including scheduling plugins, are presented in [17]. In such a way, a number of extensions is available within GSSIM including simulated architecture flexibility, advanced workload management, network simulation, application performance modeling, simulation of energy efficiency and web GUI enabling remote management and execution of simulation experiments. All these features are described in the following sections.

3.1. Simulated architecture

The main goal of GSSIM is to enable researchers to effectively perform experiments that contain simulations of distributed computing environments. Therefore, it assumes a distributed infrastructure with multiple administrative domains (called also sites in this paper) and scheduling entities. In general, GSSIM models two generic types of scheduling entities: global and local schedulers. Global scheduler, is responsible for scheduling jobs to resources that belong to different administrative domains. To this end, it must interact with multiple sites (the most common example of a site is a computing cluster under control of one of popular queueing systems, such as PBS, LSF, SGE, etc.) including retrieving information about resources, submitting jobs, or creating reservations depending on specific settings and a type of considered scheduling problem. A local scheduler is responsible for managing resources that belong to a single administrative domain (site). It retrieves tasks and reservation requests from global schedulers. GSSIM allows to build a hierarchy of local schedulers corresponding to the hierarchy of resource components over which the task may be distributed (e.g., clusters and computing nodes).

Having these two generic entities, GSSIM can be configured to model a large scope of architectural patterns. To this end, users may define for each global scheduler to which local schedulers or other global schedulers it can submit tasks and/or reservation requests. An example of defining distributed architectures with multiple workloads and Grid schedulers is presented in Fig. 3. In this example, we assume that there are two Grid schedulers, which manage their separate workloads. These schedulers have access to multiple local schedulers that can subsequently form vari-

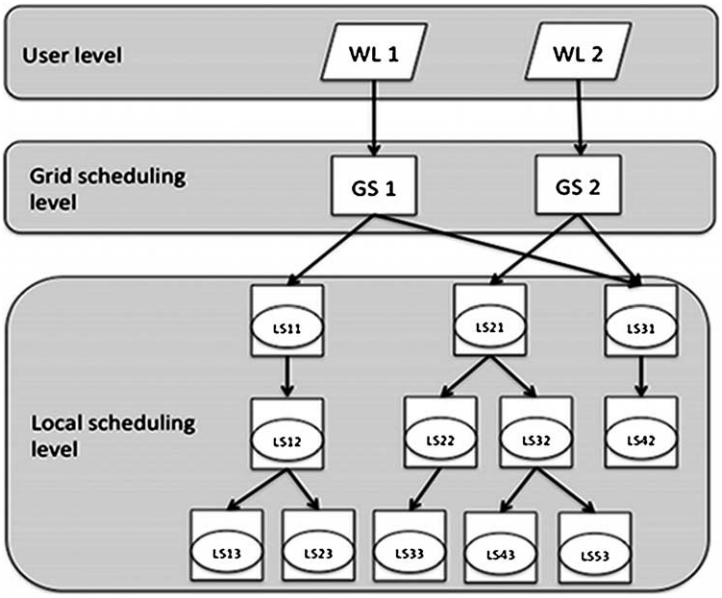

Fig. 3. Example of distributed scheduling architecture in GSSIM; WL – Workload, GS – Grid scheduler, LS – Local scheduler.

ous hierarchies. However, each Grid scheduler can access a different set of local ones. On the other hand, more than one Grid schedulers can submit tasks to a single local scheduler. This architecture is just an example, so obviously other configurations are possible, for instance, fully distributed case with a separate workload for each scheduler.

3.2. Workload management

In this section details concerning the workload structure and management are described.

3.2.1. Workload structure

Experiments performed in GSSIM require a description of jobs and tasks which will be scheduled during simulation. As a basic description, GSSIM uses files in the Standard Workload Format (SWF) [26] or its extension Grid Workload Format (GWF) [10]. In addition to the SWF file, some more detailed description of a job and task can be provided in additional XML file. Each XML file represents one job, and each job consist of multiple tasks. This form of description provides a scheduler with more detailed information about task requirements, user preferences and execution time constraints, which are unavailable in SWF/GWF files. An example of such information is a set of parameters related to execution time constraints. They express user's knowledge about task execution time and user's requirements about the earliest start and the latest end time of a task – parameters essential in scheduling with advance reservations. Other parameters define dependencies between tasks in order to build workflows.

As mentioned in Section 2.1, jobs may have various shapes and levels of complexity. Thus, scheduling strategies may have different scope and need different input data. Therefore, GSSIM distinguish several levels of information about incoming jobs. These levels and relationships between jobs and tasks are illustrated in Fig. 4.

3.2.2. Workload generator

The main purpose of the workload generator tool is to create synthetic workloads. It generates standard SWF workloads as well as additional parameters in auxiliary file (in the XML format).

All elements of the workload, mentioned in the previous section, can be described by a number of attributes, beginning from a number of tasks, task arrival time, task runtime, through task resource requirements, such as requested number of CPUs, up to user's preferences. Input parameters of the workload generator cover most of workload attributes. Each configuration parameter is described by standard statistical attributes, such as mean, minimum, maximum, standard deviation, and may have predefined probabilistic distribution, e.g., normal or constant.

One of the objectives of the workload generator is to create synthetic workloads which are similar to real

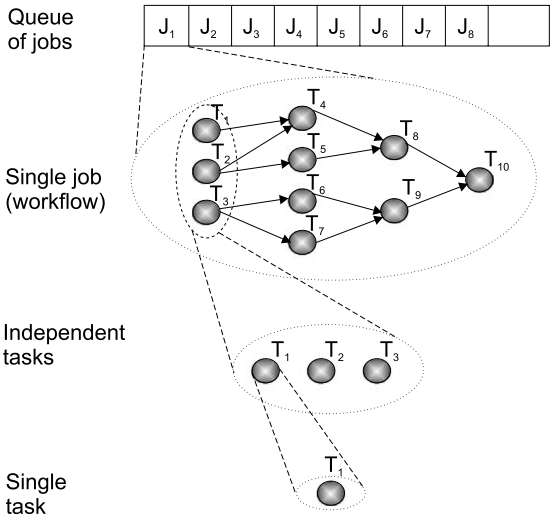

Fig. 4. Levels of information about jobs.

ones [16]. Therefore it is possible to define dependencies between any two parameters in configuration file. Exact values can be replaced by the dependency in a form of a mathematical expression which allows to use basic operators like $+$, $-$, $*$, $/$, $(,)$. For instance, dependency between a number of CPUs and memory used by jobs can be defined.

In addition to parameter dependencies, researchers can define parameters which have different probability distributions at different time periods. This feature allows to reflect the natural changes of tasks' parameter distributions at certain time or, in other words, to model daily cycles. For instance, tasks which are submitted in night hours may be longer then tasks submitted during a day time.

Another feature of the workload generator which allows to create more natural workloads is the use of multiple distributions for different parts of a workload. It is possible to define for a single parameter a number of distributions which describes only some percentage of generated values. For example, 30% of tasks may require 10 CPUs and the remaining 70% may require 5 CPUs.

3.3. Application performance modeling

GSSIM provides means to include specific application performance models during simulations. To this end, additional plugin and interface are included in the GSSIM framework. Implementation of this plugin allows researchers to introduce specific ways of calculating task execution time.

The following parameters can be applied to specify execution time of a task:

- Processor type and parameters.
- Available memory.
- Task length (number of CPU instructions needed to complete a task).
- Network parameters.
- Task requirements.
- Input data size.

Based on these parameters an estimated execution time can be calculated in various ways depending on the specific applications and scenarios.

The basic plugin available within the GSSIM release implements the most common performance model of a task, i.e. linear dependency between execution time and resource speed. According to the classification given in Section 2.1, this plugin implements a model with uniform processors. Time is calculated based on an actual task length (measured as a number of CPU operations) and CPU speed (expressed as a number of operations per second). Let us denote the former parameter as l_i, where i is a number of a given task, and the latter parameter as μ_j, where j is a number of a machine. Additionally let us assume that n_j is a number of processors of machine j allocated to this task. Then we obtain actual execution time of task i on machine j, denoted as p_{ij}, using the following simple formula:

$$p_{ij} = \frac{l_i}{\mu_j n_i}. \tag{1}$$

Of course, we can easily imagine more complex dependencies between execution time and parameters of resources and application. For instance, speed up resulting from parallelization usually is worse than linear. Therefore, instead of proportional decrease of an execution time for bigger numbers of processors one can model it using other functions, for instance of the form:

$$p_{ij} = \frac{l_i}{\mu_j \ln n_i}. \tag{2}$$

In the case of parallel and distributed applications we should also include information about available network parameters to model speed-up realistically. Other important issues include task memory requirements and machine memory limits.

Using parameters listed in this section developers can, for instance, take into account architectures of un-

derlying systems, such as multi-core processors, or virtualization overheads, and their impact on final performance of applications.

3.4. Flow-based simulation of network using advance reservation

Several simulation tools use a network model based on a packet-level network approach (see Section 2.3). This model assumed packetizing data that is sent over network into packets limited by the Maximum Transmission Unit (MTU) size. As all packets are sent over the links this approach causes a significant overhead in simulating large data transfers. For this reason, the network flow model was introduced [3]. It models data transfers as flows instead of sending large numbers of packets. Applied bandwidth sharing model uses simple min–max bandwidth fair sharing, i.e. each flow that shares a link receives an equal portion of the bandwidth.

GSSIM uses the flow networking concept because using this fluid view of network traffic the speed of simulations is largely improved by avoiding a need to packetize large network transfers as shown in [3]. Furthermore, GSSIM enhances the network flow model by additional functions that provide more complex information about network topology and by the network advance reservation functionality. These capabilities are relevant for experiments with co-allocation of various types of resources and for analysis of data management aspects in distributed computing, especially in grids and clouds. Architecture of the flow-based network simulator supporting advance reservations, which was adopted in GSSIM is illustrated in Fig. 5.

The figures shows an example of network topology that consists of nodes: routers, sites, and links between them. It also presents two components which were added to GSSIM: Network Manager and Path Computing Element. Arrows show interactions between the components. The Network Manager provides basic information about network, such as network topology, bandwidth and latency between nodes. Moreover, it is responsible for handling network reservation process including creating, canceling and modifying reservations. It updates calendars which are associated with every link. The calendar represents changes of the link bandwidth over a period of time. The second important module is the Path Computing Element (PCE), which supports Network Manager by providing two network features. PCE finds the shortest path between two nodes by adopting Dijkstra's algorithm and calculates the maximum flow between each pair of sites. It returns a list of calendars for each connection between the nodes.

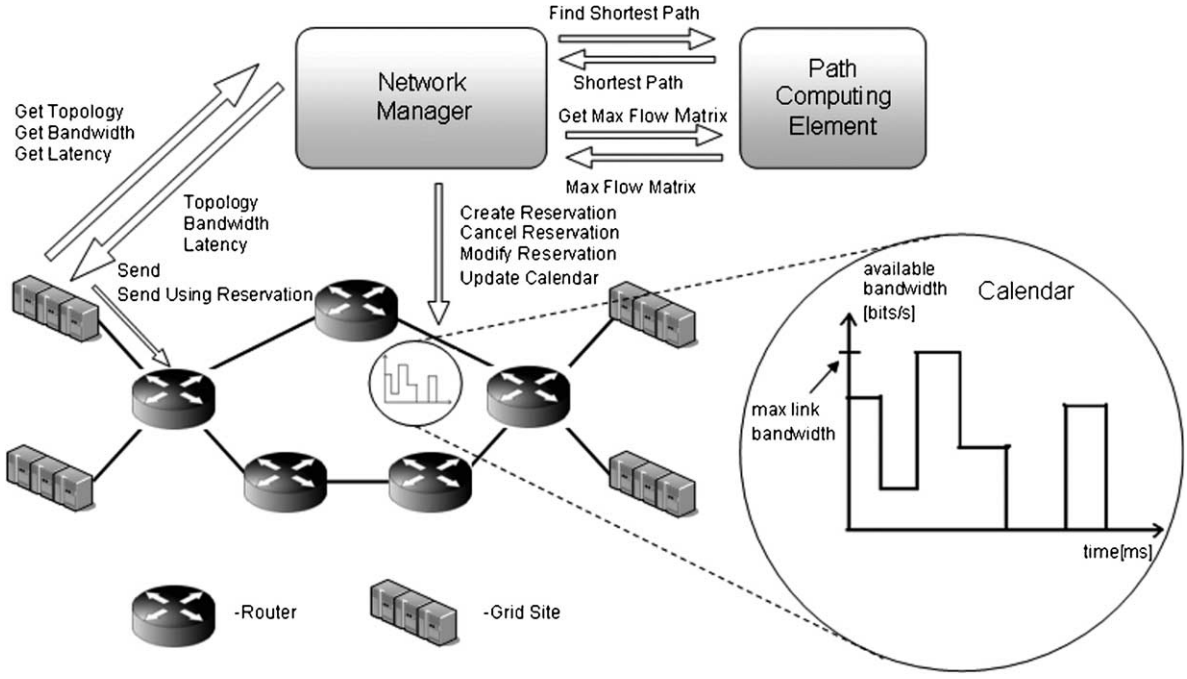

Fig. 5. Managing network advance reservations in GSSIM.

A common use cases may look as follows. A scheduler queries the Network Manager about a network topology and then about bandwidth on a given path. Then it makes a decision whether to send data in the best effort manner relying on the information it received or to send data using reservation. In the second case it is necessary to create a reservation first. The reservation guarantees constant bandwidth during sending data with the reservation period. A scheduler sends its request to the Network Manager which allows it to create a reservation on particular path or on the shortest path between given nodes with a requested bandwidth in a given period of time. It results in updating calendars related to the reserved links. Once a reservation has been created data can be sent with a proper reservation identifier. Reservation can be altered or cancel.

In order to add the advance reservation capability to the network simulation, in addition to changes of the architecture and information exchanged among components, enhancements of the flow networking model were needed. In general, duration of a network flow is a sum of latency and time needed to send data through the network. The latter is a ratio of data size to the available bandwidth. This total time can be calculated using Eqs (3)–(5) [3]. Let $LAT(f)$ be the total latency of a flow f from u to v. Then the total latency is a sum of latencies of all edges on the path that connects the source u to the destination v, i.e.

$$LAT(f) = \sum_{(u',v')\epsilon f} LAT(u', v'). \tag{3}$$

Minimal bandwidth $\min BW_{\min}(f)$ is the smallest bandwidth available on any edge on path between u and v and is given by the formula:

$$BW_{\min}(f) = \min_{(u',v')\epsilon f} \frac{BW(u', v')}{n(u', v')}, \tag{4}$$

where $n(u', v')$ is the number of active flows over link (u', v').

Now, given that SIZE(f) is a number of bytes in flow f, the total duration of network flow f can be calculated as:

$$T(f) = LAT(f) + \frac{SIZE(f)}{BW_{\min}(f)}. \tag{5}$$

Taking advance reservations into consideration Eq. (3) remains unchanged (since latency is the same) while Eqs (4) and (5) must be transformed as follows:

$$BW_{\min}(f, r_i) = \frac{BW_{\mathrm{R}}((u, v), r_i)}{n_{r_i}(u, v)}, \tag{6}$$

$$T(f, r_i) = LAT(f) + \frac{SIZE(f)}{BW_{\min}(f, r_i)}, \tag{7}$$

where r_i is a reservation of network used to transfer given data. The whole path (u, v) is considered instead of particular links (u', v') since within one reservation the same bandwidth is reserved at all links in the path. Thus $BW_{\mathrm{R}}((u, v), r_i)$ is a bandwidth allocated to reservation r_i while $BW_{\min}(f, r_i)$ is a bandwidth available for flow f using reservation r_i. In this case $n_{r_i}(u, v)$ denotes active flows within a given reservation (multiple data transfers can be performed within a single reservation). Equations above express bandwidth and duration of network flow with respect to reservation r_i. If data is transferred in a best effort manner, i.e. through non-reserved links, total time is calculated as in (5), while bandwidth is given by:

$$BW_{\min}(f) = \min_{(u',v')\epsilon f} \frac{BW(u', v') - BW_{\mathrm{R}}(u', v')}{n(u', v')}. \tag{8}$$

The use of flow networking concept with presented extensions allows GSSIM to provide efficient simulations of network including reservation capabilities.

3.5. Simulation of energy efficiency

GSSIM allows researchers to take into account the energy consumption issue in distributed computing simulations [14]. To introduce the energy consumption to a simulation environment appropriate energy consumption models must be used. The main goal of the models is to emulate the behavior of the real computing resource and the way it consumes energy. Due to reach functionality and flexible environment description, GSSIM can be used to develop new energy consumption models or to examine energy management strategies. In more detail, GSSIM provides a functionality to define energy efficiency of resources, dependency of energy consumption on resource load and specific applications, and to manage power modes of resources. The energy consumption models provided by default in GSSIM can be classified into following groups, starting from the simplest model up to the more complex ones:

Static approach is based on a static definition of resource power usage. The model calculates total amount of energy consumed by the computing resource system as a sum of energy consumed by all its components (processors, disks, power adapters, etc.). In the simplest case specific power usage values are assigned to computing nodes. More advanced versions of this approach assume definition of resource states along with corresponding power usage. Energy states are defined separately for each component of the computing resource system such as processor, memory, disk, power adapter, etc. By default, similar to processor C-States, on/off/sleep/stand-by basic states are supported. However user can define any number of new, resource specific, states. For the processor it is also possible to define frequency (voltage) levels, so called P-States, in which processor can operate with specific power usage levels. This model follows changes of resource energy states and calculates amounts of energy defined for each state.

Resource load model expands static energy state description and enhances it with realtime resource usage, most often simply the processor load. In this way it enables a dynamic estimation of power usage based on resource basic power usage and state (defined by static resource description) as well as resource load. For instance, it allows to distinguish amount of energy used by idle processor and a processor at full load. In this manner energy consumption is directly connected with energy state and describes average power usage by the resource working in a current state. For the processors which can operate on more than one frequency level, such description must be provided separately for each level.

Application specific model allows to express differences in the amount of energy required for executing various types of applications at diverse computing resources. It considers all defined system elements (processors, memory, disk, etc.), which are significant in a total energy consumption. Moreover, it also assumes that each of these components can be utilized in a different way during the experiment and thus have different impact on total energy consumption. To this end, specific characteristics of resources and applications are taken into consideration. Various approaches are possible including making the estimated power usage dependent on defined classes of applications, ratio between CPU-bound and IO-bound operations, etc. All these dependencies can be modeled in the energy profiles, special plugins used to customize estimation of energy consumptions to specific applications and hardware.

In order to model energy management in GSSIM a researcher has to perform several steps. First, a resource description has to be prepared. Developer, should specify power usage of resources. Depending on an accuracy of a model user may provide additional information about energy states which are supported by the resources, amounts of energy consumed in these states, energy consumption by specific subcomponents, or energy consumption related to resource load to calculate the total energy consumed by the resource during runtime. If high accuracy and customization to specific applications and hardware is required, a user can defined energy profiles that provide means to calculate dynamic energy consumption according to application and resource models provided. Information provided within the resource description, both static resource description and values calculated by the resource energy profile, can be used to perform advanced resource management. GSSIM provides interfaces, which allow scheduling plugins to collect detail information about computing resource components and to change their energy states. Presence of detailed resource usage information, current resource energy state description and functional energy management interface enables an implementation of energy-aware scheduling algorithms. Resource energy consumption becomes in this context an additional criterion in the scheduling process, which use various techniques to decrease energy consumption, e.g., workload consolidation, turning off unused resources, cutting down CPU frequency and others. After experiment performed using GSSIM the energy management process and efficiency of used policies can be summarized and analyzed. To this end, detailed data about each resource component state and the energy consumed by it is collected. To ensure appropriate level of details each change of the resource component energy state, each value returned by the resource energy profile is logged along with time stamp and presented in a user friendly chart form.

3.6. Remote management and execution of simulation experiments

GSSIM project is accompanied by the web interface and system that provides online access comprehensive experiment editor, remote experiment execution as well as the experiments repository. Performing experiments requires to establish the simulation environment properties first. GSSIM interface offers intuitive resource and network topology editor which

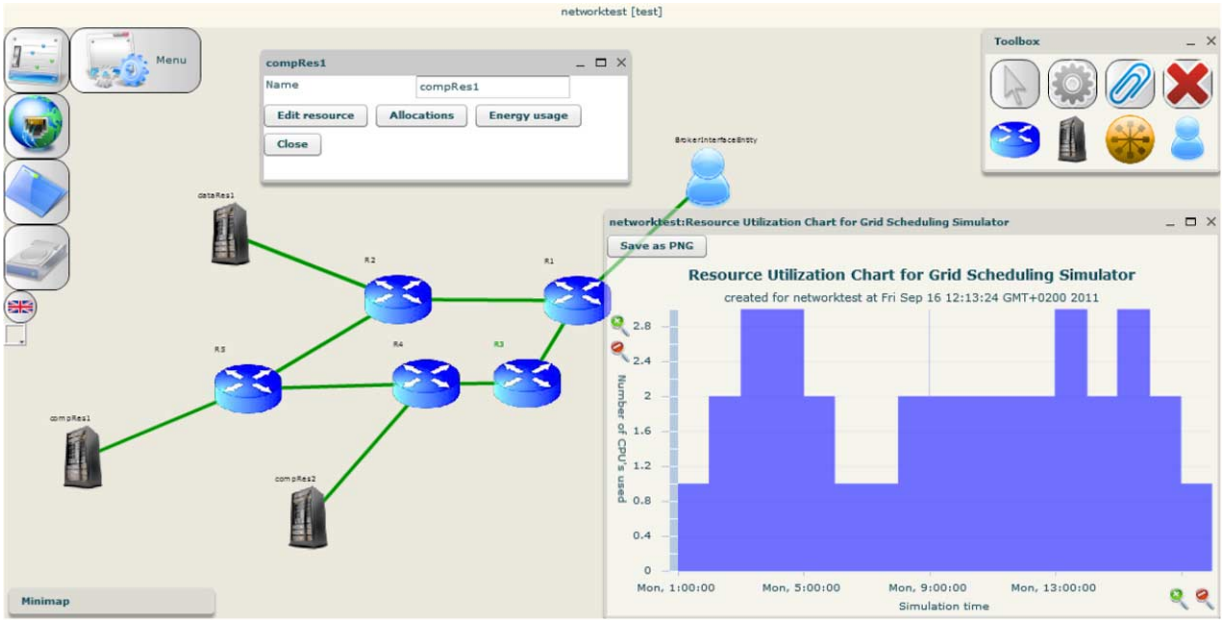

Fig. 6. GSSIM GUI. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0332.)

guides user through this stage. It supports the drag and drop paradigm and therefore relieves the user from the need of preparing the input files manually. Every resource can be easily edited by dedicated forms, which enables defining appropriate entity parameters. Web GSSIM interface includes also workload editor and generator tool. It provides all necessary means to facilitate the whole process and create the workload according to the statistical parameters specified by the user. Visualization of results is possible through interactive graphs that allow users to adjust the chart content to their requirements. Moreover, users can effortlessly customize chart size using advanced mechanism of zooming, switch between various charts and quickly view interesting details about jobs and resources. The concept of GSSIM experiment groups enables users to, first, run multiple options of one experiment at once and, afterwards, to study the impact of specific parameters on simulation results. An example of the experiment editor window with generated chart for resource utilization is presented in Fig. 6.

4. Simulation experiments

GSSIM, due to its flexibility, can be applied to a wide scope of experiments. As it was mentioned in Section 2.4, these experiments can be conducted by scientists to verify their research hypotheses, administrators to safely tune configuration of their production systems or by resource owners to assist them in making decisions concerning extensions of computing infrastructure. GSSIM has been already successfully applied to many scheduling problems, e.g., [15,16,20].

To demonstrate capabilities of the simulator we present in this section three examples of results of experiments conducted using the GSSIM framework. The first experiment uses the best effort strategies and real traces to investigate consequences of resource partitioning. In the second experiment we illustrate the use of detailed Gantt diagrams to study and compare two scheduling algorithms. The third experiment contains simulations of algorithms using advance reservation on the basis of the synthetic workload generated by GSSIM.

4.1. Study of partitioning

Proper configuration of resource management systems is a complex task performed by administrators of computing infrastructures. This configuration must ensure efficient management of workloads and high resource utilization. One of techniques used is partitioning of larger pools of resources into smaller parts. Partitioning may need introduction of two levels of workload management. Nevertheless, consequences of such decision should be evaluated a priori to avoid problems with the real system operation. The simulation envi-

ronment such as GSSIM is a perfect tool to perform such *what-if* analysis. In this section we illustrate how GSSIM can be used to study this problem based on real workload traces. We assumed the use of best effort strategies such as First Come First Serve (FCFS) and the off-line scheduling strategy Largest Size First (LSF). Thus, let us show now experimental results obtained by FCFS and LSF evaluated in the light of the following criteria: *utilization*, *waiting time* and *flow time* for a smaller fraction of the SDSC SP2 workload (jobs identified from 1 to 1000), see Table 3.

In contrast to FCFS, LSF is an example of the off-line strategy which is usually invoked periodically in a scheduling system to sort the queue according to job-size parameters. Obviously, the longer system waits to apply the LSF strategy the higher number of jobs is collected for further processing. This natural characteristic of any off-line scheduling strategy influences the overall performance of the system, especially all evaluation metrics defined based on flow and waiting time parameters as there are no scheduling decisions made during the off-line period. On the other hand, a set of collected jobs together with their requirements allows a scheduler to take into account simultaneously all job and resource characteristics to optimize its scheduling decisions. Therefore, there is always a trade-off between on-line and off-line configuration parameters which in practice are adjusted experimentally. Therefore, we decided to invoke LSF with different frequencies in another simulation tests we conducted, respectively every $p = 10$, $p = 60$, $p = 600$ and $p = 3600$ s. As presented in Table 3, FCFS strategy achieved still good results, but it was outperformed by LSF-10 and LSF-60 describing the configuration of LSF with the off-line scheduling period $p = 10$ and $p = 60$ s, respectively. All four strategies FCFS, LSF-10, LSF-60 and LSF-600 reached the same level of utilization $U = 0.545$, however the total waiting time was reduced by 2% in the case of LSF-10 and 3% in the case of LSF-60. As expected, much longer invocation periods of the LSF strategy affected significantly the mean flow time, and corresponding total waiting time, increased up to 81,621 and 73,449, respectively. At the same time, LSF-3600 strategy reduced utilization down to the level of 49%, however, because of the high peak load selected from the SDSC SP2 workload for this experiment, further improvements for this objective would be difficult to achieve.

To perform more comprehensive simulation studies we introduced hierarchical scheduling structures with two local queues involved. First, we partitioned computing resources into two sets based on additional descriptions and comments to real workloads at [26]. In practice, partitioning techniques are often used by local administrators giving them the possibility of assigning end-users or resources to various queues depending on the monitored workload and past system behaviors. Therefore, for further simulation experiments, all created partitions and attached computing resources were provided to the Grid-level over a certain number of local queues. Let us first describe hierarchical structures we used during the next experimental tests. A number of computing resources reported for SDSC SP2 was 128, and first we created two partitions providing the access over two queues to 2×64 computing resources. However, we had to modify a bit the original SDSC SP2 workload by reducing the requested computing resources from 128 to 64 for jobs identified as: 86, 91, 95, 144, 171 (end-user 92) and 208, 742, 746, 750, 920, 967 (end-user 147). To evaluate hierarchical scheduling structures, Random and Load Balancing strategies were used to assign jobs at the grid-level to local queues, while FCFS and LSF-600 strategies were applied for local queues.

As we can observe in Table 4, Partitioning the SDSC SP2 system into two parts resulted in a lower utilization of computing resource. As expected, such a hierarchical scheduling structure did not outperform the performance of the original structure where only one queue was used together with 128 computing resources. Even the reduced job sizes of a few jobs in the analyzed workload did not help to improve the total waiting time and mean flow time in the hierarchical system with two local queues. However, compar-

Table 3
Performance of local scheduling strategies: FCFS and LSF applied for the SDSC SP2 workload (jobs 1–1000)

Queues × Resources	Grid-level	Local-level	Utilization	Flow time (s)	Waiting time (s)
1×128	FCFS	FCFS	0.545	16,137	7965
1×128	FCFS	LSF-10	0.545	15,972	7800
1×128	FCFS	LSF-60	0.545	15,896	7724
1×128	FCFS	LSF-600	0.545	18,127	9955
1×128	FCFS	LSF-3600	0.494	81,621	73,449

Table 4
Performance of two-level hierarchical scheduling structures for the SDSC SP2 workload (jobs 1–1000)

Queues × Resources	Grid-level	Local-level	Utilization	Flow time (s)	Waiting time (s)
2 × 64	Random	FCFS	0.517	41,625	33,366
2 × 64	Random	LSF-600	0.502	42,895	34,636
2 × 64	Load balancing	FCFS	0.53	33,479	25,220
2 × 64	Load balancing	LSF-600	0.532	34,988	26,728

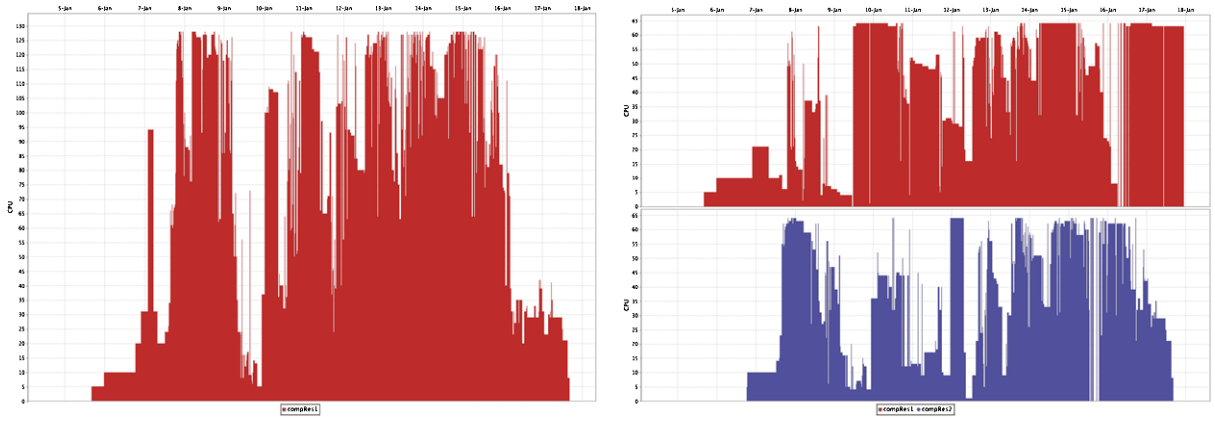

Fig. 7. Comparison of utilization generated for the original single-queue and two-queue partitioned SDSC SP2 system. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0332.)

ing strategies at the Grid-level, one should note that Load Balancing algorithm significantly outperformed the Random approach with respect to all evaluated criteria. It allows to increase utilization of resources by 6%, while the flow time and waiting time were reduced by 19 and 24%, respectively. Thus, these results confirmed the importance of the resource allocation phase on a grid. Moreover, the configuration of Load Balancing and LSF-600 procedures turned out to be better than Load Balancing and FCFS strategies with respect to utilization criterion. Hence, it may be worthwhile to introduce more advances scheduling, i.e. off-line scheduling, so that a scheduler can take into account simultaneously more task and resource characteristics to better optimize schedules. Generated by GSSIM utilization of resources for different hierarchical configurations (single queue with FCFS and LSF-600 and two-queue partitioned with Load Balancing and LSF-600 strategies) used in simulation experiments is presented in Fig. 7. More details of this experiment can be found in [21].

GSSIM simulation environment allowed us to perform this study without additional effort required to establish specific configuration of the real infrastructure. Use of GSSIM to simulate production environments may save a lot of work and inconvenience for users of the system.

4.2. Comparison of algorithms

This section contains results of a comparison of two scheduling policies: on-line (a single task at once) vs. off-line (scheduling a set of tasks at once on a available resource pool). In the former case a scheduler can allocate resources (specific time slot) for a single task in the head of the queue. It takes tasks one-by-one using FCFS policy, query for a slot for this task and, if query is successful, task is allocated and executed. Another approach assumes that a scheduler can select a task to execute from a set of tasks in the queue. In this case a scheduler can check particular time slots against requirements of various tasks. One of the simplest methods that take advantage of this approach is the Graham's algorithm [9] which we adopted for this comparison.

In Fig. 8 one can easily see that using Graham's algorithm the scheduler is able to submit certain tasks earlier because for each slot it checks tasks until it finds one that can be allocated to this slot. Two exemplary regions with tasks allocated earlier are marked in the figure. Generally, this advantage of the Graham

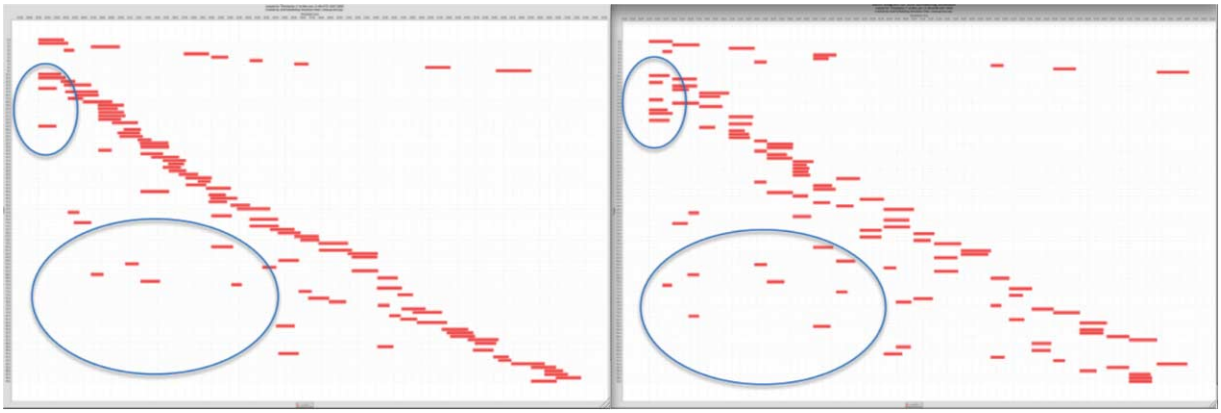

Fig. 8. Comparison of schedules obtained by FCFS (left) and Graham (right) policies. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0332.)

approach leads to better resource utilization and task-related metrics such as mean flow time. Detailed values of these metrics for both approaches and their dependency on a number of tasks are obviously available in statistics generated by GSSIM. It was easy to see that while for small number of tasks differences are small (due to substantial amount of free resources at the beginning), for greater number of tasks the Graham policy demonstrates its advantage both in terms of mean flow time and resource utilization.

More details of this experiment can be found in [24].

The results presented above are just simple example of more detailed analysis which is available in GSSIM. By browsing details of Gantt charts generated for jobs and/or reservations one may analyze phenomena that occur in the scheduling process more deeply than just based on general statistical values. More details on using GSSIM Gantt charts to analyze schedules can be found in [18].

4.3. Configuration of advance reservation

Some of resource management systems support the advance reservation functionality, e.g., [27,28]. Nevertheless, advance reservation may significantly deteriorate efficiency of the whole system. Therefore, a proper configuration of this aspect of the resource management system is very important. One of parameters that are of particular importance is a length of a reservation time slot. This length defines a minimum time period for which resources can be reserved in the system. Thus, it determines a granularity of reservations. For instance, Platform LSF [27] allows reservations in 10 min time slots (i.e., *fixed-length* time slots) while SGE [28] does not impose a minimal reservation length (referred as *variable-length* time slot in the sequel). Generally,fixed-length time slots improve performance of searching and allocation algorithms as the list of slots does not depend on a number of reservations in the system. Of course, increasing a length of a time slot leads to further performance improvements. On the other hand, long fixed-length slots result in low resource utilization. Therefore, finding an appropriate trade-off between resource utilization and performance of algorithms is an important issue.

In order to study how performance of the particular methods depends on parameters of the tasks 16 workloads diverse in terms of task sizes and lengths were prepared. We assumed that neither the earliest start times nor deadlines were defined, i.e. available resource were searched within period from current time to infinity. Resource consisted from 64 identical processors.

We studied the impact of slot lengths on quality of results, namely, resource utilization, makespan, and mean flow time. We compare the variable-length slots approach with two versions of the fixed-length slots approach: with short (10 min) and long (1 h) slot length. Resource utilization obtained using these methods is compared in Fig. 9. It is easy to see that the longer length of a fixed slot the bigger the number of “gaps” that leads to lower resource utilization.

The results presented above are confirmed in Fig. 10 where our expectations concerning resource utilization are confirmed by precise values. They are complemented by values of makespan and mean flow time. More details of this experiment can be found in [24].

An administrator looking at the presented resource utilization charts as well as performance statistics may choose an appropriate time slot length for the system in question.

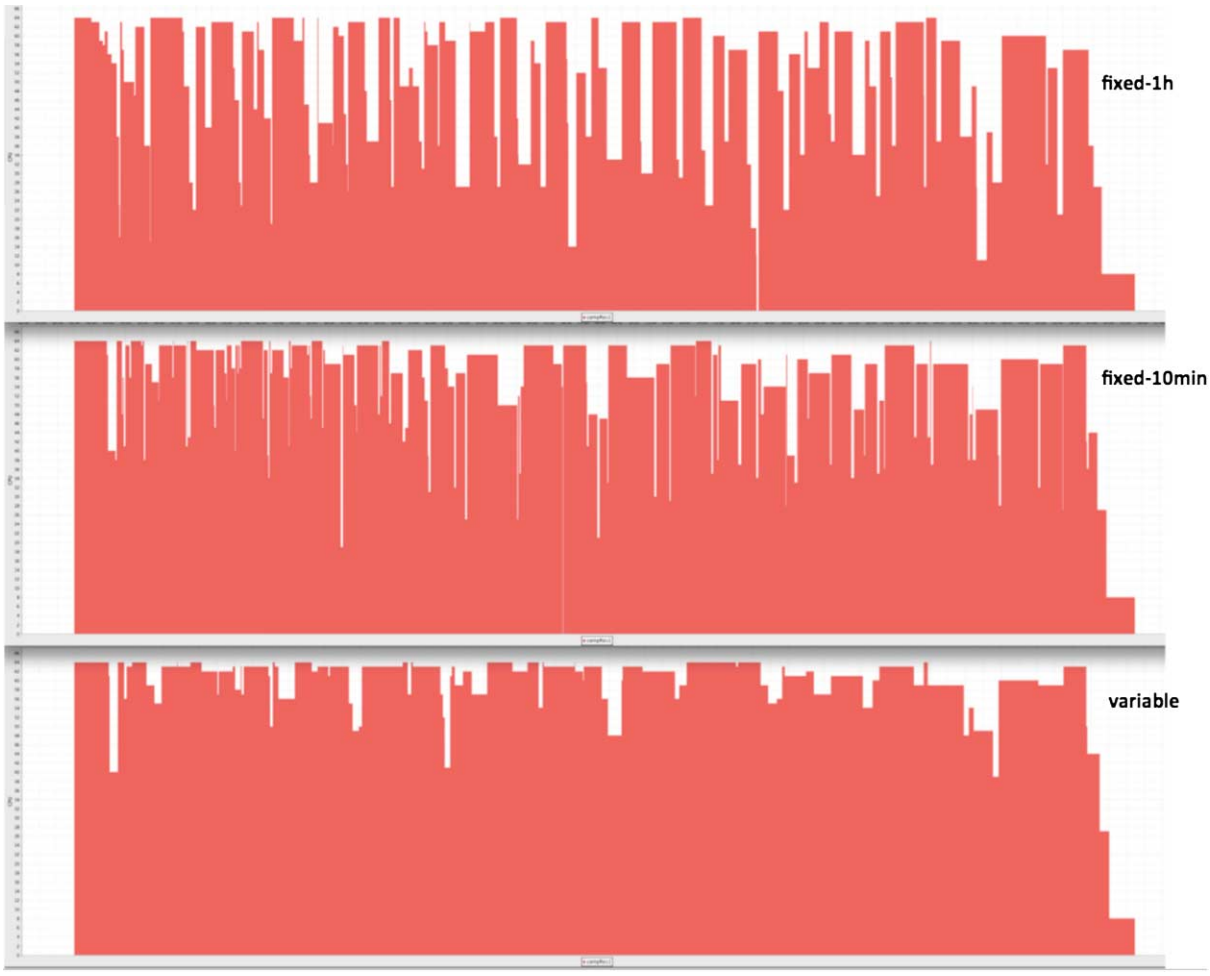

Fig. 9. Resource utilization for the fixed- and variable-length slots approach. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0332.)

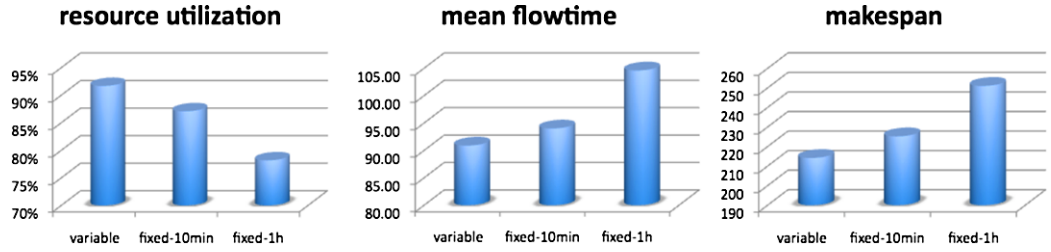

Fig. 10. Resource utilization, makespan and mean flow time for variable- and fixed-length slots approaches. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0332.)

5. Conclusion and future work

In this paper we presented GSSIM – a simulation framework that addresses the issues relevant to distributed computing experiments. In particular, GSSIM aims at facilitating, automating, and accelerating the process of preparation, execution, and analysis of experiments. We compared GSSIM with other known simulators based on a classification of distributed computing simulator features proposed in this paper. After analysis of this comparison we concluded that GSSIM delivers several functionalities which are not, to the

best of our knowledge, provided by any other distributed computing simulation tool available, namely: efficient network modeling including advance reservation capability, the possibility of adding customized application performance models related to both execution time and power usage, advanced modeling of various events including network failures and security issues, and the possibility of managing and executing simulations remotely in the cloud. GSSIM also contains a flexible workload generation tool allowing any number of jobs with sophisticated requirements to be created. Moreover, obtained experimental results can be analyzed using a fine-grained visualization of schedules and resource utilization. We have demonstrated that GSSIM supports a variety of scheduling problems and scenarios common in distributed computing environments, with specific examples of a few different experiments using queue-based strategies and real traces, as well as testing algorithms with advance reservation on synthetic workloads.

To enable sharing of workloads, algorithms and results, we have proposed the GSSIM portal [30], from where researchers may download various synthetic workloads, resource descriptions, scheduling plugins, and results. Users can even manage and run the whole experiments through the GSSIM web interface. This portal complements other known websites related to this area as it provides a repository of synthetic workloads and scheduling plugins as well as the online services for workloads generation and remote execution of experiments.

As GSSIM is a highly customizable and extendable framework which enables a wide range of possible future works. Therefore, we will focus on customization of simulations for specific scenarios, systems, and applications. For example, we will aim at more detail modeling of thermal effects caused by application load and virtualization overheads of specific cloud platforms. This will be achieved by development of new plugins and preparation of workloads and resource descriptions, which will be shared via the GSSIM portal. In this way the online simulation service will be constantly extended in order to meet the requirements of the distributed computing community.

Acknowledgements

The research presented in this paper was partially supported by a grant from Polish National Science Centre under award number 5790/B/T02/2010/38.

References

[1] W.H. Bell, D.G. Cameron, A.P. Millar, L. Capozza, K. Stockinger and F. Zini, Optorsim: a grid simulator for studying dynamic data replication strategies, in: *Proceedings of IJHPCA*, SAGE Publications, London, 2003, pp. 403–416.

[2] J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt and J. Węglarz, *Handbook of Scheduling: From Theory to Applications*, Springer-Verlag, Berlin, 2007.

[3] J. Broberg and R. Buyya, Flow networking in grid simulations, in: *Grid Computing: Infrastructure, Services, and Applications*, L. Wang, ed., CRC Press, Boca Raton, FL, 2009, pp. 389–404.

[4] R. Buyya and M. Murshed, 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)* **14** (2002), 1175–1220.

[5] D.G. Cameron, R. Carvajal-Schiaffino, J. Ferguson, P. Millar, C. Nicholson, K. Stockinger and F. Zini, OptorSim: a simulation tool for scheduling and replica optimisation in data grids, in: *Proceedings of Computing in High Energy Physics*, Interlaken, Switzerland, 2004.

[6] H. Casanova, A. Legrand and M. Quinson, SimGrid: a generic framework for large-scale distributed experimentations, in: *Proceedings of the 10th IEEE International Conference on Computer Modelling and Simulation*, IEEE Computer Society, Los Alamitos, CA, 2008, pp. 126–131.

[7] Federica, http://www.fp7-federica.eu/.

[8] D.G. Feitelson and L. Rudolph, Toward convergence in job schedulers for parallel supercomputers, in: *Job Scheduling Strategies for Parallel Processing*, D.G. Feitelson and L. Rudolph, eds, Springer-Verlag, London, 1996, pp. 1–26.

[9] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling theory: a survey, *Annals of Discrete Mathematics* **5** (1997), 287–326.

[10] Grid workloads archive, http://gwa.ewi.tudelft.nl/.

[11] Y. Huang, A. Brocco, M. Courant, B. Hirsbrunner and P. Kuonen, MaGate simulator: a simulation environment for a decentralized grid scheduler, in: *Proceedings of the 8th International Symposium on Advanced Parallel Processing Technologies*, Springer-Verlag, Berlin, 2009, pp. 273–287.

[12] A. Iosup, O.O. Sonmez and D.H.J. Epema, DGSim: comparing grid resource management architectures through trace-based simulation, in: *Proceedings of the 14th international Euro-Par Conference on Parallel Processing*, Springer-Verlag, Berlin, 2008, pp. 13–25.

[13] D. Klusáček and H. Rudová, Alea 2 – job scheduling simulator, in: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ICST, Brussels, Belgium, 2010, pp. 1–10.

[14] M. Krystek, K. Kurowski, A. Oleksiak and W. Piątek, Energy-aware simulations with GSSIM, in: *Proceedings of the COST Action IC0804 on Energy Efficiency in Large Scale Distributed Systems*, Cost Office, Toulouse, 2010, pp. 55–58.

[15] M. Krystek, K. Kurowski, A. Oleksiak and K. Rządca, Comparison of centralized and decentralized scheduling algorithms using GSSIM simulation environment, in: *CoreGrid Integration Workshop*, Springer-Verlag, New York, 2008, pp. 185–196.

[16] K. Kurowski, J. Nabrzyski, A. Oleksiak and J. Węglarz, A multicriteria approach to two-level hierarchy scheduling in grids, *Journal of Scheduling* **11** (2008), 371–379.

[17] K. Kurowski, J. Nabrzyski, A. Oleksiak and J. Węglarz, GSSIM – grid scheduling simulator, *Computational Methods in Science and Technology* **13** (2007), 121–129.

[18] K. Kurowski, A. Oleksiak, W. Piątek and J. Węglarz, Hierarchical scheduling strategies for parallel tasks and advance reservations in grids, *Journal of Scheduling* (2011), 1–20.

[19] K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak and J. Pukacki, Improving grid level throughput using job migration and rescheduling techniques in GRMS, in: *Scientific Programming*, IOS Press, Amsterdam, 2004, pp. 263–273.

[20] K. Kurowski, A. Oleksiak and J. Węglarz, Multicriteria, multiuser scheduling in grids with advance reservation, *Journal of Scheduling* **13** (2010), 493–508.

[21] K. Kurowski, Multicriteria resource management in grid environments with dynamic descriptions of jobs and resources, PhD thesis, Poznan University of Technology, 2009.

[22] MetaCentrum, http://www.fi.muni.cz/~xklusac/index.php?page=meta2009.

[23] M. Mika, G. Waligóra and J. Węglarz, Modelling and solving grid resource allocation problem with network resources for workflow applications, *Journal of Scheduling* **14** (2011), 291–306.

[24] A. Oleksiak, Multicriteria job scheduling in grids using prediction and advance resource reservation mechanisms, PhD thesis, Poznan University of Technology, 2009.

[25] S. Ostermann, K. Plankensteiner and R. Prodan, Using a new event-based simulation framework for investigating different resource provisioning methods in clouds, *Scientific Programming Journal* **19**(2,3) (2011), 161–178.

[26] Parallel workload archive, http://www.cs.huji.ac.il/labs/parallel/workload/.

[27] Platform LSF, http://www.platform.com/.

[28] SGE, http://www.sun.com/software/sge/.

[29] A. Sulistio, C.S. Yeo and R. Buyya, A taxonomy of computer-based simulation and its mapping to parallel and distributed systems simulation tools, *International Journal of Software: Practice and Experience* **34** (2004), 653–673.

[30] The Grid Scheduling Simulations Portal, http://www.gssim.org.

[31] J. Węglarz, J. Józefowska, M. Mika and G. Waligóra, Project scheduling with finite or infinite number of activity processing modes: a survey, *European Journal of Operational Research* **208** (2011), 177–205.