Alea 2 – Job Scheduling Simulator

Dalibor Klusáček and Hana Rudová
Faculty of Informatics, Masaryk University
Botanická 68a
Brno, Czech Republic
{xklusac,hanka}@fi.muni.cz

ABSTRACT

This work describes the Grid and cluster scheduling simulator Alea 2 designed for study, testing and evaluation of various job scheduling techniques. This event-based simulator is able to deal with common problems related to the job scheduling like the heterogeneity of jobs, resources, and the dynamic runtime changes such as the arrivals of new jobs or the resource failures and restarts. The Alea 2 is based on the popular GridSim toolkit [31] and represents a major extension of the Alea simulator, developed in 2007 [16]. The extension covers both improved design, extended functionality as well as the improved scalability and the higher simulation speed. Finally, new visualization interface was introduced into the simulator. The main part of the simulator is a complex scheduler which incorporates several common scheduling algorithms working either on the queue or the schedule (plan) based principle. Additional data structures are used to maintain information about the resource status, the objective functions and for collection and visualization of the simulation results. Many typical objectives such as the machine usage, the average slowdown or the average response time are included. The paper concludes with an example of the Alea 2 execution using a real-life workload, discussing also the scalability of the simulator.

Categories and Subject Descriptors

D.2.8 [**SIMULATION AND MODELING**]: Types of Simulation—*Discrete event*

General Terms

Simulation, Scheduling, Visualization

Keywords

Grid, cluster, scheduling, simulation, GridSim

1. INTRODUCTION

In the recent years large computing clusters, computing centers and Grids have become more and more available to the scientific and commercial community. Efficient job scheduling in these large, dynamic and heterogeneous systems is important but difficult task. Many researchers are getting involved in developing new efficient algorithms that would be suitable either for general systems or for specific institution or company. Clearly, newly proposed algorithms must be heavily tested and evaluated before they are applied in the real systems. Due to many reasons, such as the cost of resources, the reliability, the varying background load or the dynamic behavior of components, experimental evaluation cannot be mostly performed in the real systems. To obtain reliable results, many simulations with various setups must be performed using the same and controllable conditions that simulate different real-life scenarios. This is often unreachable in the real Grid.

For this purpose many simulators have been developed. If properly designed, such simulators are very useful since different setups and different data sets can be used to evaluate existing or proposed solutions. While for some purposes an ad-hoc simulator is sufficient, there are also general Grid and cluster simulators allowing to simulate various scenarios and problems. Most of them are available as toolkits that have to be carefully modified and extended before they are suitable for the researcher's goals. The amount of such necessary work is usually quite large if the simulation outputs should be complex and reliable.

In this work we present an extension to the well known GridSim simulation toolkit. It is a Grid scheduling simulator called *Alea 2* which represents "ready to use" centralized scheduling system allowing to apply and compare various scheduling algorithms similar to those used in the production systems such as PBSpro [14], LSF [35] or CCS [12]. The solution consists of the scheduler entity and other supporting classes which extend the original basic functionality of the GridSim. The main benefit of our solution is that the Alea 2 allows immediate testing by inclusion of several popular and widely used scheduling algorithms such as FCFS, EDF [24], EASY Backfilling [29], EDF-Backfilling [36], etc. Beside the queue-based algorithms, it also enables the use of algorithms that construct the schedule — sometimes referred to as the scheduling plan [12]. The research on the schedule-based techniques have become quite popular in the recent years. In CCS system [12] backfill-like policy is used to generate schedule while GORBA scheduler [30] uses evolutionary algorithms to optimize the schedule. In [4], genetic algorithms are used for the Grid scheduling while [34, 1, 28]

SIMUTools 2010 March 15–19, Torremolinos, Malaga, Spain.
Copyright 2010 ICST, ISBN 78-963-9799-87-5.

propose several variants of local search-based methods for scheduling in the Grids and heterogeneous computer environments. Also, our own research is oriented towards the proposal of schedule-based techniques [18, 19], therefore we have included them into the simulator together with the previously mentioned queue-based algorithms. Current release of the Alea 2 also contains workload parsers that can read popular trace formats such as the Grid Workloads Format (GWF) and the Standard Workloads Format (SWF), thus allowing an immediate use of the simulator.

This paper is organized as follows. First, the related work is discussed. Next, the problem characteristics related to the simulation capabilities of the Alea 2 are described. Section 4 presents the design of the Alea 2 and its main features. We also show the scalability and the functionality of the simulator through several experiments using the largest traces from the Parallel Workloads Archive and the complex real-life workload from the Czech national Grid infrastructure MetaCentrum [17]. Finally the future work is discussed.

2. RELATED WORK

There are numerous simulators and toolkits that provide various functionality for simulations of the clusters, network and Grid environments.

We start with the *MicroGrid* [25] which is rather an emulator than a simulator. It can be used for systematic study of the dynamic behavior of applications, middleware, resources, and networks. The MicroGrid uses the Globus Toolkit 2.2 API for its execution which allows to precisely emulate GTK 2.2 based systems — however such systems are outdated since the current version 4.2.1 uses different model based on the concept of web services.

The *Bricks* [32] simulates various scheduling schemes on a typical high-performance global computing systems. The Bricks can simulate various behaviors of global computing systems, especially the behavior of networks and resource scheduling algorithms. Moreover, its modular design allows to incorporate different scheduling algorithms, and it also allows incorporation of existing global computing components via its foreign interface.

The *BeoSim* [15] has been implemented for the purpose of studying multi-site parallel job scheduling algorithms in the context of a multi-cluster computational Grid. The Beosim can be driven either by synthetic workload or real workload. It also provides Java based visualization tool.

The *SimGrid* [22] is a *C* based simulator used for the simulation and development of distributed applications in heterogeneous and distributed environment. The SimGrid solves different problems using different programming environments that constitutes different paradigms. The SimGrid's MSG tool is widely used for the basic evaluation and simulation of scheduling algorithms while other tools such as the GRAS and the SMPI are used for developing and study of real applications. The SimDag provides functionalities to simulate parallel task scheduling with DAG (Direct Acyclic Graphs) workflow models.

The *Simbatch* [3], based on the SimGrid's MSG, allows to evaluate scheduling algorithms for batch schedulers. Neither the SimGrid nor the Simbatch offer integrated visualization output. However, both simulators may generate a trace that can be lately visualized through the Pajé [6] or the ViTE [5] visualization tools.

The *SimBOINC* [20] is another SimGrid based simulator. It is designed to simulate heterogeneous and volatile desktop grids and volunteer computing systems. The SimBOINC simulates a client-server platform where multiple clients request work from a central server. The goal of this project is a testing of new scheduling strategies in the BOINC (Berkeley Open Infrastructure for Network Computing), and in other desktop and volunteer systems. The characteristics of the client such as the speed, the availability of the workload or the network availability can all be specified in the simulation inputs.

While the SimGrid, the Simbatch and the SimBOINC are all based on the *C* programming language, all following simulators are based on the popular Java language [26]. While Java is usually less efficient than *C*, it allows easy development and effortless portability.

The *Monarc 2* [8] is a simulation framework whose aim is to provide a design and optimization tool for large scale distributed computing systems, with a focus on the forthcoming LHC (Large Hadron Collider) experiments at CERN. Although it incorporates simple scheduling module, the main goal of the Monarc 2 is to provide a realistic simulation of distributed computing systems, customized for specific physics data processing, and to offer a flexible and dynamic environment for the performance evaluation of a range of possible data processing architectures. The Monarc 2 extends the obsolete Monarc simulator [23], by improving its flexibility and performance.

The *GridSim* [31] is a flexible, modular and universal Grid simulation toolkit with a very good documentation. It is written in Java on top of an event simulation library called *SimJava* [13]. Thanks to the Java language, the GridSim is a platform independent toolkit. The GridSim provides functionality to simulate the basic Grid environment and its behavior by providing simple implementations of common entities such as the computational resources or the users. It also allows to simulate simple jobs, the network topology, the data storage and other useful features. Provided implementations represent the base functionality and it is necessary to extend them when performing simulations with more complex requirements. This can been done by the implementation of new Java classes inheriting from the existing GridSim classes, which is also the case of the Alea 2 described in this paper.

The GridSim is used by various researchers in their simulations. There exists decentralized scheduler [2] implemented in an obsolete version of the GridSim, but it does not support dynamic behavior of the system and it is not capable of simulating network topology or other recent features. The reason is the incompatibility between older and recent GridSim versions.

The *Grid Scheduling Simulator (GSSIM)* [21] based on the GridSim toolkit has been developed for several years and became publicly available in 2009. It should provide an easy to use Grid scheduling framework for enabling simulations of a wide range of scheduling algorithms in multi-level, heterogeneous Grid infrastructures. However, the GSSIM encounters some problems such as slow execution, weak scalability and poor visualization outputs. Moreover, the GSSIM is not compatible with the standard GridSim releases (including the latest GridSim 5) since it uses its own branch based on a modified version 4. Still, if GSSIM is optimized and made compatible with the GridSim, it will provide an interesting alternative, supporting more features than our simulator.

The *Alea* [16] has been developed since 2007. However, over the time many core improvements have been done concerning the design, the scalability and the functionality. Several new scheduling algorithms and objective functions were included as well as the support of additional job and machine characteristics. The new Alea 2 now also provides complex visualization tool which supports an export of simulation results into several bitmap formats. The simulation speed and the simulator's scalability have been significantly improved through the newly developed or redesigned classes. This covers a new memory-efficient job loader and a redesigned job allocation policy, which speeds up the whole simulation (see Section 5). Moreover, the support of standardized workloads formats has been included as well as the simulation of machine failures using the real-life failure traces. All these features are closely discussed in Section 4.

To conclude this section, there are several different simulation toolkits that cover various aspects of Grid and cluster simulations. Sadly — based on our experience — several of them are not now usable[1] either due to the incompatibility (MicroGrid), major reconstruction (SimBOINC), or due to the authors' decisions (Bricks, BeoSim).

3. PROBLEM CHARACTERISTICS

In this section we briefly describe the type of problems which may be simulated using the Alea 2. We focus on the problem of Grid and cluster scheduling that can be formulated by defining the characteristics of machines, jobs and by optimization criteria. In addition, we also discuss the behavior of the system with respect to the dynamic changes occurring over time.

Simulated system is composed of one or more computer clusters, that are managed by one centralized scheduler. One cluster is composed of several machines. So far, we expect that machines within one cluster have the same parameters. Those are the number of CPUs, size of the main memory (RAM) and CPU speed. Moreover, each machine may have additional parameters that closely specify its additional properties. These properties typically describe the architecture of the machine (Opteron, Xeon, . . .), available software licenses (Matlab, Gaussian, . . .), operating system (Debian, SUSE, . . .), maximum time limit for job execution (e.g., 2 hours, 24 hours, 1 month), network interface parameters (10Gb/s, Infiniband, . . .), available file system (nfs, afs, . . .) or the owner of the machine (Masaryk University, Charles University, . . .). All machines within a cluster use the Space Sharing processor allocation policy which allows the parallel execution of several jobs at the cluster when the total amount of requested CPUs is less or equal to the number of CPUs of the cluster. Therefore several machines within the same cluster can be co-allocated to process a given parallel job. On the other hand, machines belonging to different clusters can not be co-allocated (used for execution of the same parallel job).

Job represents user's application. Job may require one (sequential) or more CPUs (parallel). Also the duration and the arrival time are specified. There are no precedence constraints among jobs and we consider neither preemptions of the jobs nor migrations from one machine to another. Additional information may closely specify the job's characteristics. Those are the job deadline, the maximum time limit for execution, the required machine architecture, the requested software licenses, the operating system, the network type, etc. These additional information are very important since they shift the simulation's results closer to the reality as we have shown in our previous work on real-life complex data sets [17]. The goal of the scheduler is to meet all these requirements. Moreover, scheduler should also reflect the quality of the generated solution. For this purpose, several objective functions are supported, that cover both users' and system administrators' expectations. High machine usage, high throughput and good performance of the scheduler are usually very important for the system administrator. Therefore, the avg. machine usage [9], the avg. weighted machine usage [33], the makespan [34], the avg. number of waiting jobs and the avg. runtime of the scheduling algorithm [19] are used to measure the performance of the scheduler. Users' requirements are represented by the avg. slowdown [9], the avg. weighted slowdown [9], the number of delayed jobs [19] and the avg. tardiness [16]. There are some criteria that are useful for both the users and the administrators such as the avg. response time [9], the avg. weighted response time [9] and the avg. wait time [9].

Another important parameter is the dynamic behavior of the simulated system. Job arrivals and completions appear during the time, change the load of the system and request appropriate reactions of the scheduler. Another factor are the dynamic changes on the machines. As the time is running, machines may become unavailable due to the failures or the upgrades while other machines may appear as a result of restart or extension of current cluster's size. Again, such situations have to be properly handled by the scheduler.

4. ALEA 2 OVERVIEW

Same as the GridSim, the Alea 2 is an event-based modular simulator, composed of independent entities which implements the desired simulation functionality (see Figure 1). It consists of the centralized scheduler, the grid resource(s) with the local job allocation policy, the job loader, the machine and failure loader and additional classes responsible for the simulation setup, the visualization and the generation of simulation output. By now, the Grid users are not directly simulated but the job loader entity can be used as a parent class for the future implementation of the Grid user. Simulator's behavior is driven by the event-based message passing protocol. For each simulated event — such as the job arrival or the job completion — one message defining this event is created. It contains the identifier of the message recipient, the type of the event, the time when the event will occur and the message data. In case of, e.g., job arrival such message would look like this: (scheduler_ID; job_arrival_event; job_arrival_time; job_description). The simulator is fully compatible with the latest GridSim 5.0[2] release since no changes were made in the GridSim package itself. All extensions were made by implementing child classes which extend the standard GridSim (parent) classes. Similarly, easy extension of current functionality is possible thanks to the object oriented paradigm used by the GridSim and the Alea 2. In the following text all important extensions on top of the GridSim package are mentioned and explained.

The simulation is initialized by the `ExperimentSetup` class which creates instances of the scheduler, the job and machine

[1]In October 2009.

[2]GridSim 5.0 was released in September 2009.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8722
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8722

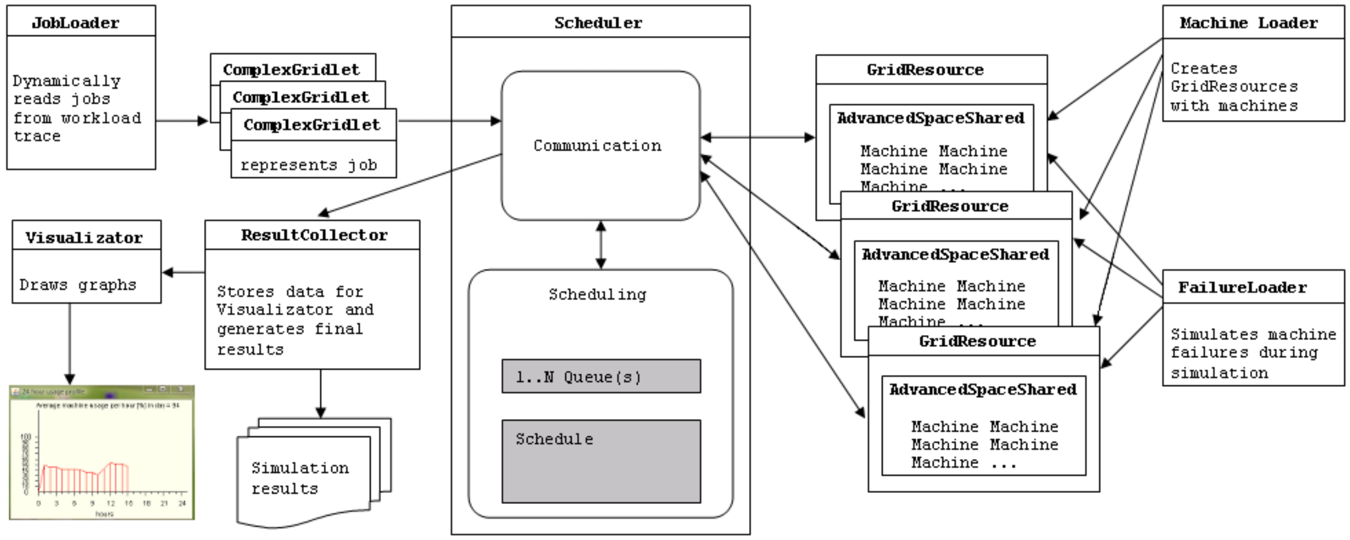


Figure 1: Main parts of the Alea 2 simulator.

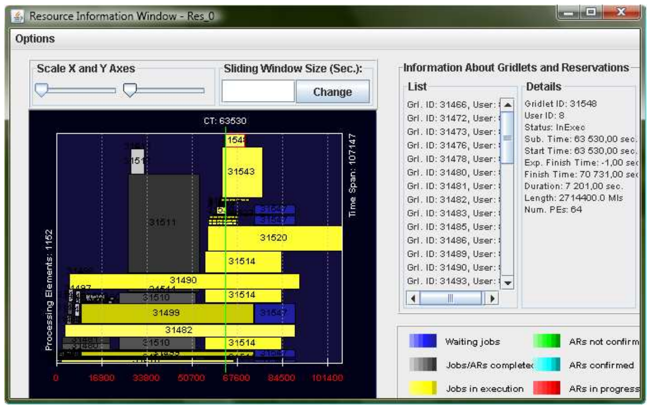
loader, the failure loader and other entities as required by the standard GridSim. The `MachineLoader` entity performs the initialization of the simulated computing environment. It reads the data describing the machines from a file and creates Grid resources accordingly. GridSim itself does not provide such functionality and machine parameters must be "hardcoded" in the source java file. Our solution allows to change machines' parameters without the re-compilation of the whole program.

The `JobLoader` reads the file containing the job descriptions and creates jobs' instances dynamically over the time. The `JobLoader` supports several trace formats including the Grid Workloads Format (GWF) of the Grid Workloads Archive[3] and the Standard Workloads Format (SWF) of the Parallel Workloads Archive[4]. When the simulation time is equal to the job submission time the `JobLoader` sends the job to the scheduler. The `JobLoader` reads only one job at a time to limit the required memory space, and to allow the use of very large workload traces. This approach was taken since the original GridSim's `Workload` entity reads all data at once which results in simulation fails for large workloads due to the Java's Out-Of-Memory error. The job itself is represented by the instance of the `ComplexGridlet` class. The GridSim provides only trivial implementation of a job in its `Gridlet` class. The `ComplexGridlet` extends this class, allowing to simulate more realistic scenarios where each job may require additional properties such as the specific amount of available memory or the specific machine parameters, and other real-life based constraints as was discussed in Section 3.

The `FailureLoader` reads the file containing descriptions of machine failures. Once the simulation time reaches the failure start time, the appropriate machine is set to be failed, killing all jobs being currently executed on that machine. When the failure period passes the machine is restarted.

In the GridSim, the Grid resource is represented by the `GridResource` instance. It contains one or more machines that constitute, e.g., a computer cluster. Such resource is managed by the local scheduling policy. Current GridSim 5.0 supports several policies based on space-shared and time-shared paradigm. Unfortunately none of these policies support the execution of parallel jobs and the simulation of machine failures at the same time. Moreover, the co-allocation of several machines is also not available. Therefore, new allocation policy called `AdvancedSpaceShared` was developed for the Alea 2 based on the GridSim's `SpaceShared` policy. It allows to execute both sequential and parallel jobs on the specified number of CPUs using the space-shared processor allocation policy. It enables to simulate more realistic scenarios involving the parallel jobs as well as the simulations of machine failures. In addition, it includes more efficient implementation of the message passing which allows significant improvements in the simulation speed (see Section 5).

The newly developed `Visualizator` class generates the simulation's graphical output. The Gridsim itself enables visualizations displaying the process of allocating jobs onto the machines over time as can be seen in Figure 2. The `Visu-`



Grl. ID: 31466, User:
Grl. ID: 31472, User:
Grl. ID: 31473, User:
Grl. ID: 31476, User:
Grl. ID: 31478, User:
Grl. ID: 31480, User:
Grl. ID: 31481, User:
Grl. ID: 31482, User:
Grl. ID: 31483, User:
Grl. ID: 31485, User:
Grl. ID: 31486, User:
Grl. ID: 31489, User:
Grl. ID: 31490, User:
Grl. ID: 31493, User:

Gridlet ID: 31548
User ID: 8
Status: InExec
Sub. Time: 63 530,00 sec.
Start Time: 63 530,00 sec.
Exp. Finish Time: -1,00 sec
Finish Time: 70 731,00 sec
Duration: 7 201,00 sec.
Length: 2714400.0 MIs
Num. PEs: 64

Figure 2: Visualization of the job allocation procedure as provided by the GridSim.

[3]`http://gwa.ewi.tudelft.nl`

[4]`http://www.cs.huji.ac.il/labs/parallel/workload/`

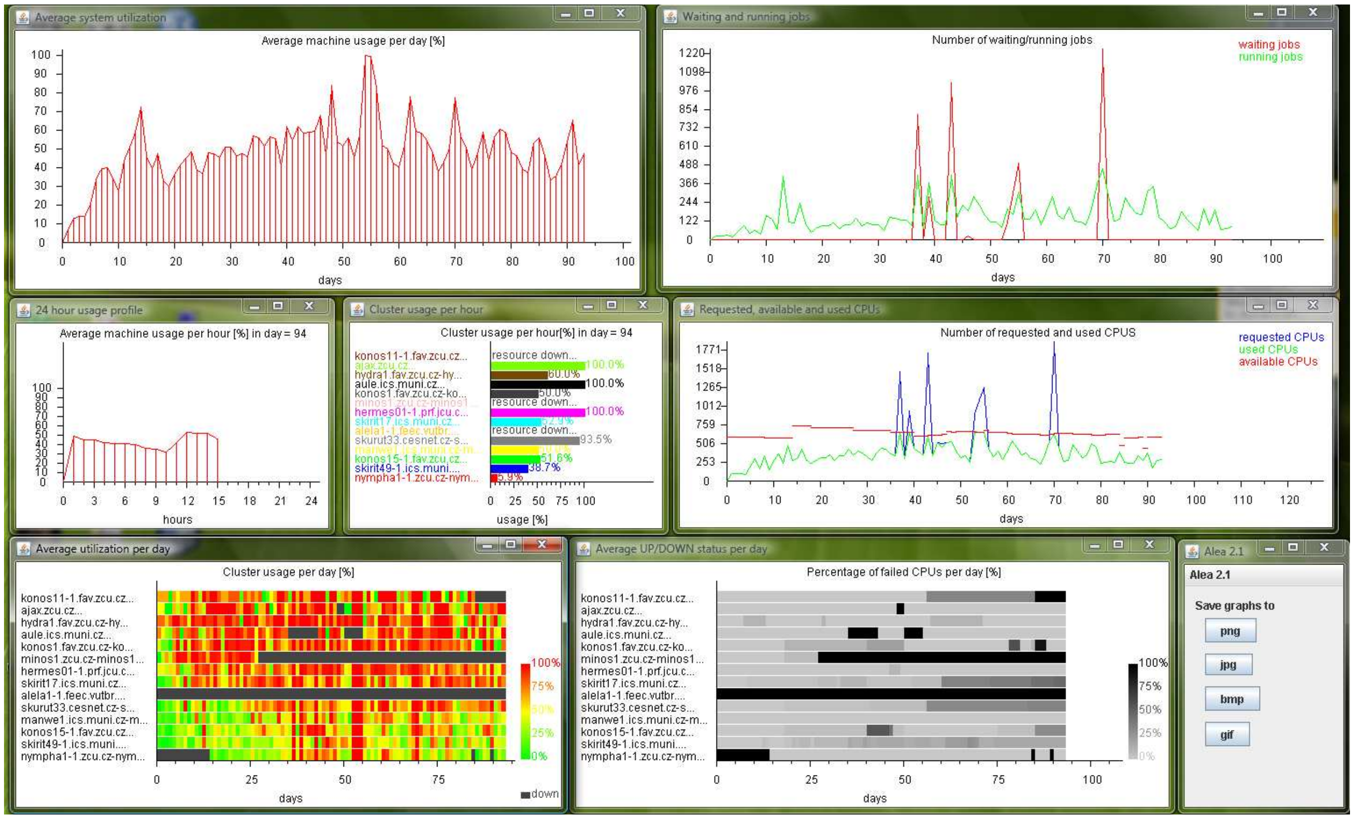

Average system utilization
Average machine usage per day [%]
days
Waiting and running jobs
Number of waiting/running jobs
waiting jobs
running jobs
days
24 hour usage profile
Average machine usage per hour [%] in day = 94
hours
Cluster usage per hour
Cluster usage per hour[%] in day = 94
usage [%]
Requested, available and used CPUs
Number of requested and used CPUS
requested CPUs
used CPUs
available CPUs
days
Average utilization per day
Cluster usage per day [%]
days
down
Average UP/DOWN status per day
Percentage of failed CPUs per day [%]
days
Alea 2.1
Save graphs to
png
jpg
bmp
gif

Figure 3: Visualization interface of the Alea 2 during the simulation.

`alizator` extends this functionality by displaying additional information useful for tuning and debugging of scheduling algorithms. So far, several outputs covering different objectives are supported and displayed. Those are the average overall utilization of resources, the cluster utilization, the number of waiting and running jobs and the number of requested, utilized and available CPUs. Beside that, also the percentage of failed and running CPUs per cluster can be displayed. The `Visualizator` may work in two different fashions. In the first case, the visualization is generated continuously as the simulation proceeds (see Figure 3). In the second case, the graphs are generated when the simulation is finished, using the simulation results as an input. Results are continuously collected by the `ResultCollector`. When the simulation completes, the `ResultCollector` stores them into *csv* files that can be easily used as an input for other tools (Calc, Excel, Spreadsheet, etc.) and it also saves generated graphs into a preferred bitmap file (png, jpg, bmp, gif).

The key part of the Alea 2 is the `Scheduler` entity, which is described in the following section.

4.1 Scheduler Entity

The `Scheduler` is the main part of the Alea 2. Its behavior is driven by events and corresponding messages. Using them, the `Scheduler` communicates with the `JobLoader` (job arrivals), the `GridResources` (job submission/completion and failure detection) and with the `ResultCollector` (periodical result collection). It is responsible for performing scheduling decisions according to the selected scheduling policy. The `Scheduler` was designed as a modular, extensible entity composed of three main parts (see Figure 4) which are discussed in the following text.

The first part stores dynamic information concerning the Grid resources (see Figure 4 bottom right). For each `GridResource`, one `ResourceInfo` object is created that holds up-to-date information regarding the current resource status. It stores information about jobs currently in execution, about jobs that are planned for execution (if the schedule is being constructed) and it implements various functions that help to compute or predict various values, e.g., the next free slot available for specific job, etc.

The second part is responsible for the communication with the remaining simulation entities (see Figure 4 top). It accepts incoming messages (events) and reacts accordingly. Typically, the `Scheduler` receives newly incoming job from the `JobLoader`. It takes the incoming job and places it into the queue or schedule according to the applied scheduling algorithm. Next, new scheduling round is performed and an attempt to submit jobs present in the queue or schedule is performed. If some resource is available and a suitable job is selected, it is submitted to the resource where it will be executed. Moreover, appropriate scheduler's `ResourceInfo` object is updated according to the new situation. Once some job is completed, it is returned to the `Scheduler` and the `ResourceInfo` object is updated as a result of the new state. Similar update is performed when some machine fails or restarts. Next, a new scheduling round is started. The cycle finishes when no new job arrivals appear and all submitted jobs have been completed. Then the simulation ends

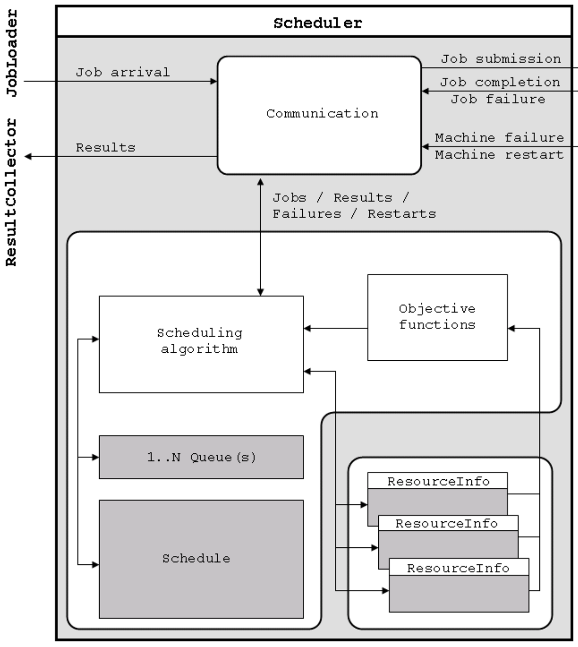


Figure 4: Main parts of the Scheduler entity.

and the results are stored into the output files.

Last part of the `Scheduler` contains implementations of several popular and widely used scheduling algorithms. Since the recent research in the area of Grid and cluster scheduling focuses on both queue and schedule based techniques we support both of them. Concerning the queue-based techniques following algorithms are implemented: *First Come First Served (FCFS)*, *Earliest Deadline First (EDF)* [24], *EASY Backfilling (EASY-BF)* [29], *EDF-Backfilling (EDF-BF)* [36] and a *PBSpro-like (PBS)* multi-queue and priority-based scheduling algorithm [14]. Schedule-based techniques use a schedule — instead of a queue(s) — to store the jobs. In this case, each job is placed into the schedule upon its arrival which defines its expected start time, expected completion time and the target machine(s). The use of the schedule allows to use advanced scheduling and optimization algorithms [27] such as the local search-based methods [11, 10]. These techniques are represented by the *Earliest Suitable Gap (ESG)* algorithm and a *Local Search (LS)* based optimization routine [18, 19]. ESG is the schedule's analogy of the queue-based backfilling approach. It was inspired by the algorithm used in the CCS system [12]. Several objective functions (see Section 3) are supported which can be used for decision making or optimization. Of course, new objective functions and new scheduling algorithms may be easily added into the `Scheduler`, using the provided data structures and interfaces.

During the simulation, the `Scheduler` is capable of collecting various data such as the number of waiting and running jobs, current machine utilization, and the information related to all other objectives which were discussed in Section 3. Once the simulation is finished, output files containing these data are generated. Moreover, selected objectives can be used as an input for either the "on the fly" or the "post mortem" visualization provided by the `Visualizator` graphical tool.

4.2 Extensibility

The Alea 2 can be easily extended thanks to the adopted object oriented paradigm. The simulator is modular, meaning that different functionality is implemented in different classes. Also, the crucial `Scheduler` class is divided into separate parts. Thus, a new scheduling algorithm or a new objective function can be added through the extension or the modification of the existing classes. Similarly if, a new job property or a new job type is requested, only the `ComplexGridlet` shall be extended or modified, leaving the other classes intact. Therefore, all changes are encapsulated and the development is straightforward.

5. EVALUATION

In this section we show the performance of the Alea 2 simulator through several experiments. First, we show the simulation involving complex real-life data set from the Czech national Grid infrastructure MetaCentrum. Next we demonstrate the speed and the scalability of the simulator with respect to the GridSim based GSSIM simulator and the original GridSim toolkit. All experiments were performed on Intel Core2 Duo 2.4GHz PC with 2 GB of RAM. Unless otherwise indicated, the JVM (Java Virtual Machine) was limited by 1 GB of available RAM.

In the first experiment, we use the complex data from the Czech national Grid infrastructure MetaCentrum[5] to demonstrate the simulation capability of the Alea 2. These data allow us to perform very realistic simulations, involving all characteristics discussed in Section 3. For example, we use precise information concerning the machine parameters, the information about machine failures and restarts, or the specific job requirements concerning the target machine properties [17]. The experiment involved 103,656 jobs that were originally executed during the first five months of the year 2009 on 14 clusters having 806 CPUs. Through the experiment, we have compared three different scheduling algorithms: FCFS, EASY-BF and schedule-based ESG. Figure 5 presents graphs depicting the average machine usage per cluster (left) and the number of waiting and running jobs per day (right) as were generated by the Alea 2 during the experiment. These graphs nicely demonstrate major differences among the algorithms. Concerning the machine usage — as expected — FCFS generates very poor results. FCFS is not able to utilize available resources when the first job in the queue requires some specific and currently unavailable machine(s). At this point, other "more flexible" jobs in the queue can be executed increasing the machine utilization. This is the main goal of the EASY-BF algorithm. As we can see, EASY-BF is able to increase the machine usage by using the backfilling approach. Still, EASY-BF does not allow to delay the execution start of the first job in the queue, which restricts it from making more aggressive decisions that would increase the utilization even more. Schedule-based ESG algorithm does not apply such restrictions and thanks to the application of a more efficient schedule-based approach it produces the best results.

In case of the second criteria, similar reasons as in the previous example caused that FCFS is not able to schedule jobs fluently, generating huge peak of waiting jobs during the time. For the same reason, the resulting makespan is also much higher than in the remaining algorithms (by 60 days).

[5]`http://meta.cesnet.cz`

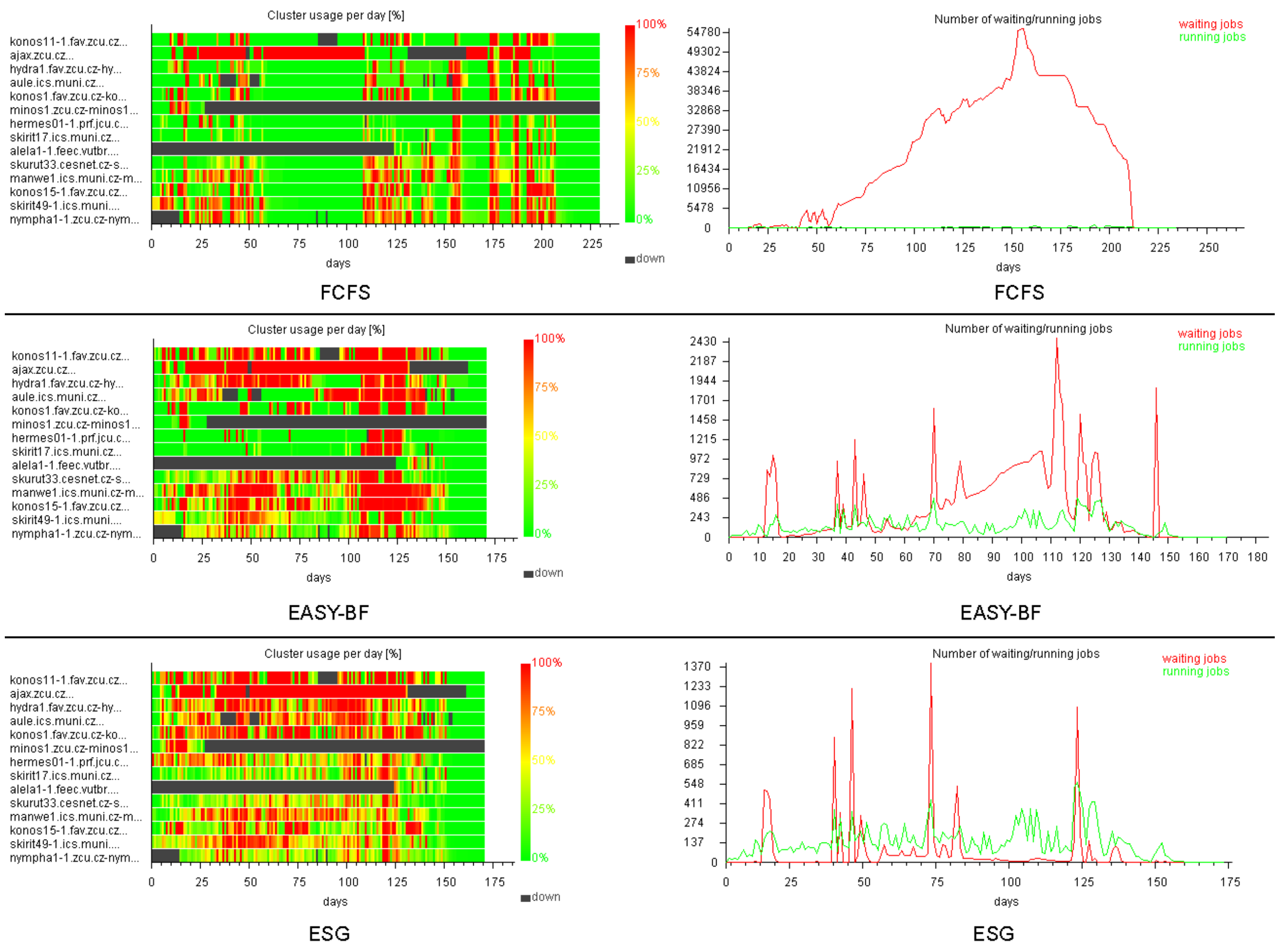


Figure 5: Comparison of FCFS, EASY-BF and ESG using complex data set.

EASY-BF is capable of a higher resource utilization and reduction of the number of waiting jobs through the time. ESG again produces the best results. As can be seen, these and several other graphical outputs such as those presented in Figure 3 help the user to understand and compare the scheduling process of different scheduling algorithms.

Our second experiment focused on the comparison of the Alea 2 and the GridSim based GSSIM simulator through an experimental execution. We could not include the GridSim itself into this experiment since it does not provide implementations of any scheduling algorithms. In this case, three different criteria were taken into account — the size of the experiment, the amount of time needed to execute the experiment as well as the RAM requirements of the simulator. The setup was identical for both simulators, using FCFS as a scheduling algorithm and the SDSC Blue Horizon workload file from the PWA as an input. Originally, we intended to compare the simulators using the largest available workload SHARCNET, but — due to the GSSIM's parser setup — GSSIM was not able to parse the workload file correctly. Therefore, we chose the second largest SDSC Blue Horizon workload containing 243,314 jobs. However, the GSSIM's execution always finished with an Out-Of-Memory error due to the given JVM's RAM limit being 1 GB. Therefore, we decreased the number of simulated jobs to an acceptable level. Through several experiments, 15,000 jobs was found to be the acceptable number. Using this setup, we compared the GSSIM with the Alea 2.

There was a large difference concerning the RAM requirements. The Alea 2 (see Figure 6(a)) required less than 4 MB of memory, while the GSSIM needed all dedicated memory (1,024 MB) as is shown in Figure 6(b). The execution time presented in logarithmic scale is available in Figure 6(c). The GSSIM uses three steps during simulation. In the first step the simulator initializes itself by reading the workload, next the scheduling is performed and finally the results are generated and stored. The Alea 2 has significantly shorter first phase since the workload is not read during this phase but it is processed "on-the-fly" during the whole scheduling period. The total execution time was measured as a sum of the initialization time, the scheduling time and the time needed for the output generation. Figure 6(c) shows that the Alea 2 outperforms the GSSIM in all phases. The third phase where results are generated was critical for GSSIM. Figure 6(d) shows the execution time of the scheduling phase with respect to the number of finished jobs. Again, the Alea 2 is faster than the GSSIM during the whole simulation. Surprisingly, the GSSIM's graph plugin failed to gener-

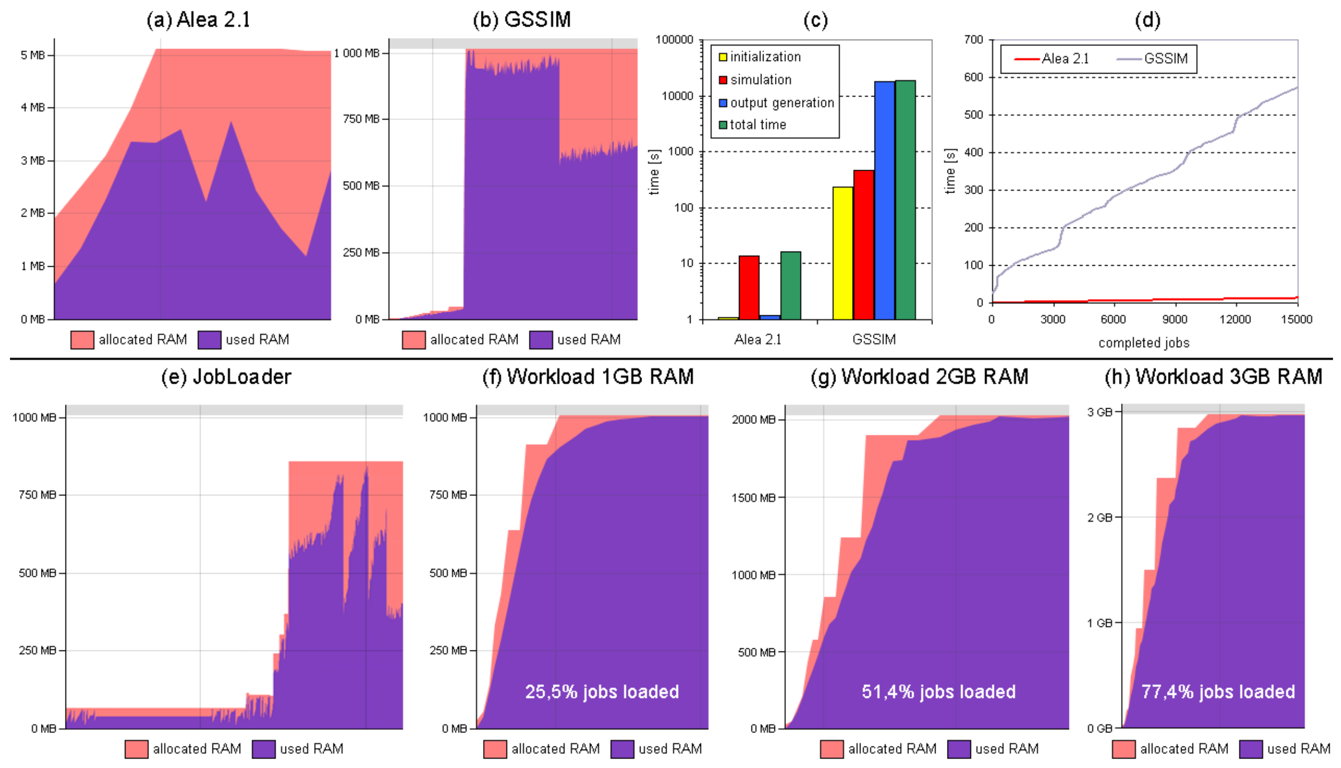


Figure 6: Experimental results for the Alea 2, the GSSIM and the GridSim.

ate any graphical output for every setup that involved more than 100 jobs due to the Java's Out-Of-Memory error. In general terms, the Alea 2 proved to be faster and less memory demanding than the GSSIM. We discussed our findings with the GSSIM developers who informed us about finishing the preparations of a new GSSIM version that should solve the problems we have encountered.

In the last experiment, the new Alea's `JobLoader` entity and the new `AdvancedSpaceShared` policy are studied using the large real-life workload traces. They are compared with the original GridSim's solutions since they are critical for a good scalability and performance of the simulator. As we already mentioned in Section 4, the GridSim provides the `Workload` class to read job descriptions from the workload file. However, the `Workload` reads all jobs *before* the simulation starts, therefore all jobs have to be stored in the RAM. For large data sets and common computers this may be a problem. On the other hand, our solution using the `JobLoader` reads and creates jobs "on the fly" as simulation proceeds, meaning that only currently running and waiting jobs are stored in the RAM at any moment. It allows us to maintain memory requirements in a decent level. In our experiment, we have compared the memory requirements of the Alea 2 with our `JobLoader` solution and with the original GridSim's `Workload` solution. We used the largest SHARCNET data set from the PWA. It contains information about 1,195,242 jobs that were executed on 6,828 CPUs at the Shared Hierarchical Academic Research Computing Network (SHARCNET) in Ontario (Canada) mostly during the year 2006. In this experiment the amount of RAM consumed by the Java Virtual Machine (JVM) was analyzed. The results for the Alea 2 using the `JobLoader` are shown in Figure 6(e). Clearly, during the whole simulation 850 MB of RAM is needed at most. On the other hand, when the JVM's limit is 1 GB, and the GridSim's `Workload` solution is used, it was not possible to load more than 25.5% of jobs as can be seen in Figure 6(f). When an additional RAM module was added into our testing computer and the JVM's memory was increased to either 2 GB or 3 GB, it was not possible to read all jobs (see Figure 6(g) and (h), respectively) and the simulated execution did not even started. Only 51.4% of jobs were loaded for the 2 GB limit, while the percentage of loaded jobs was equal to 77.4% for the 3 GB limit. Clearly the `JobLoader`'s "on the fly" approach is more suitable and more memory efficient for the large data sets.

Finally, we have compared the simulation speed of the Alea 2 for our `AdvancedSpaceShared` policy and the original GridSim's `SpaceShared` policy using the same data set. For the `SpaceShared`, the more jobs are executed the slower the simulation is. We managed to eliminate this issue by implementation of a more efficient message passing model in the `AdvancedSpaceShared` policy. Figure 7 demonstrates the difference between the original `SpaceShared` and the new `AdvancedSpaceShared` policy. In this experiment, the number of successfully completed jobs in one day (86,400 seconds) was computed. Our solution simulates all 1,195,242 jobs in just 125 minutes, while the original solution slows down, finishing only 96,000 jobs within 24 hours.

Beside the previous experiments, there are also other known results concerning the GridSim scalability [7]. Basically the limitation of the GridSim is based on the applied thread model which limits the number of communicating entities to approximately 10,000. Since the Alea 2 uses the GridSim, the same limitations applies for it too. We are aware

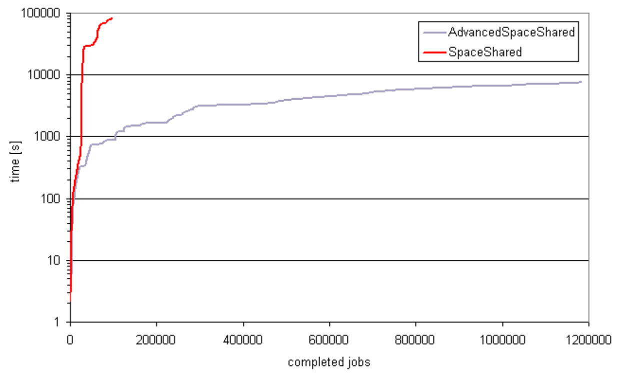


Figure 7: Simulation time with the SpaceShared and the AdvancedSpaceShared policy.

that for some types of simulations such limitation may be severe. However, the Alea 2 was designed to simulate scheduler's behavior and not the behavior of thousands Grid users. From this point of view, the ability to simulate large number of jobs and machines is necessary. Job is not a problem since it is a simple object, not using a thread. On the other hand `GridResource` representing one computer cluster requires four threads. Still, in the real world there are usually dozens or at most hundreds of clusters which fits fine within the GridSim limitations. It is necessary to notice that a machine within a `GridResource` is a simple object — not a thread — thus hundreds of machines placed within one `GridResource` require no additional thread.

6. CONCLUSION AND FUTURE WORK

We presented the platform independent Alea 2 scheduling simulator which is an extension of the popular GridSim toolkit. The goal of the simulator is to offer an easy way for the researchers to study, modify and extend various scheduling algorithms. Unlike the GSSIM or the solution described in [2], the Alea 2 stays fully compatible with the original GridSim, which is very important for the future development. It was shown that it significantly outperforms the GridSim based GSSIM simulator by means of simulation speed and scalability. Moreover, the newly developed `JobLoader` and the `AdvancedSpaceShared` solutions replacing the original GridSim's `Workload` and the `SpaceShared` solutions lead to better performance covering both the scalability (`JobLoader`) and the simulation speed (`AdvancedSpaceShared`).

Provided functionality covers typical researchers' requirements involving "ready to use" simulator that includes full implementations of common scheduling algorithms and supports common objective functions. Moreover, the visualization interface allows faster debugging and tuning of the studied algorithms as well as direct export of the simulation results into the bitmap and the *csv* files that can be easily used later. The Alea 2 has been successfully used in our work which focuses on the area of cluster and Grid scheduling and was able to simulate scheduling under different setups involving various data sets and objective functions [16, 18, 17, 19]. We also received positive feedback from several researchers from around the world who found using it very helpful.

The Alea 2, including the sources and full documentation, can be downloaded from `http://www.fi.muni.cz/~xklusac/alea`.

The Alea 2 currently supports centralized scheduling approach. We plan its extention to allow decentralized and multi-level scheduling, which is common for the large scale Grids. Also, newly proposed or adopted scheduling algorithms will be made available with the future simulator's releases. Last but not least, we plan to offer some of our solutions to become a part of the future GridSim's distributions.

7. ACKNOWLEDGMENTS

This work was kindly supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research intent No. 0021622419 and by the Grant Agency of the Czech Republic with grant No. 201/07/0205. We would like to thank to MetaCentrum supported by the research intent No. MSM6383917201 for providing us access to the workload data. The SHARCNET workload trace was graciously provided by John Morton and Clayton Chrusch and the workload log from the SDSC Blue Horizon was kindly provided by Travis Earheart and Nancy Wilkins-Diehr.

8. REFERENCES

[1] A. Abraham, H. Liu, C. Grosan, and F. Xhafa. Nature inspired meta-heuristics for grid scheduling: Single and multi-objective optimization approaches. In *Metaheuristics for Scheduling in Distributed Computing Environments*, volume 146 of *Studies in Computational Intelligence*, pages 247–272. Springer, 2008.

[2] D. Abramson, R. Buyya, M. Murshed, and S. Venugopal. Scheduling parameter sweep applications on global Grids: A deadline and budget constrained cost-time optimisation algorithm. *International Journal of Software: Practice and Experience (SPE)*, 35(5):491–512, 2005.

[3] Y. Caniou and J. S. Gay. Simbatch: An API for simulating and predicting the performance of parallel resources managed by batch systems. In *Euro-Par 2008 Workshops - Parallel Processing*, volume 5415 of *LNCS*, pages 223–234. Springer, 2009.

[4] J. Carretero and F. Xhafa. Using genetic algorithms for scheduling jobs in large scale grid applications. *Journal of Technological and Economic Development – A Research Journal of Vilnius Gediminas Technical University*, 12(1):11–17, 2006.

[5] K. Coulomb, M. Faverge, J. Jazeix, O. Lagrasse, J. Marcoueille, P. Noisette, A. Redondy, and C. Vuchener. Visual trace explorer (ViTE), October 2009. `http://vite.gforge.inria.fr/`.

[6] J. C. de Kergommeaux, B. de Oliveira Stein, and B. P.E. Pajé, an interactive visualization tool for tuning multi-threaded parallel applications. *Parallel Computing*, 26(10):1253–1274, 2000.

[7] W. Depoorter, N. Moor, K. Vanmechelen, and J. Broeckhove. Scalability of Grid simulators: An evaluation. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, volume 5168 of *LNCS*, pages 544–553. Springer, 2008.

[8] C. Dobre, F. Pop, and V. Cristea. A simulation framework for dependable distributed systems. In

Proceedings of the 2008 International Conference on Parallel Processing - Workshops, pages 181–187. IEEE, 2008.

[9] C. Ernemann, V. Hamscher, and R. Yahyapour. Benefits of global Grid computing for job scheduling. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 374–379. IEEE, 2004.

[10] F. W. Glover and G. A. Kochenberger, editors. *Handbook of metaheuristics*. Kluwer, 2003.

[11] H. H. Hoos and T. Stützle. *Stochastic Local Search Foundations and Applications*. Elsevier, 2005.

[12] M. Hovestadt, O. Kao, A. Keller, and A. Streit. Scheduling in HPC resource management systems: Queuing vs. planning. In *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *LNCS*, pages 1–20. Springer, 2003.

[13] F. Howell and R. McNab. Simjava: A discrete event simulation library for Java. In *International Conference on Web-Based Modeling and Simulation*, pages 51–56. Society for Computer Simulation International (SCS), 1998.

[14] J. P. Jones. *PBS Professional 7, administrator guide.* Altair, April 2005.

[15] W. M. Jones, W. B. Ligon, III, L. W. Pang, and D. Stanzione. Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. *The Journal of Supercomputing*, 34(2):135–163, 2005.

[16] D. Klusáček, L. Matyska, and H. Rudová. Alea – Grid scheduling simulation environment. In *7th International Conference on Parallel Processing and Applied Mathematics (PPAM 2007)*, volume 4967 of *LNCS*, pages 1029–1038. Springer, 2008.

[17] D. Klusáček and H. Rudová. Complex real-life data sets in Grid simulations. In *Cracow Grid Workshop 2009 Abstracts (CGW'09)*, 2009.

[18] D. Klusáček and H. Rudová. Efficient grid scheduling through the incremental schedule-based approach. *Computational Intelligence: An International Journal*, 2010. To appear.

[19] D. Klusáček, H. Rudová, R. Baraglia, M. Pasquali, and G. Capannini. Comparison of multi-criteria scheduling techniques. In *Grid Computing Achievements and Prospects*, pages 173–184. Springer, 2008.

[20] D. Kondo. SimBOINC: A simulator for desktop Grids and volunteer computing systems, October 2009. `http://simboinc.gforge.inria.fr/`.

[21] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz. Grid scheduling simulations with GSSIM. In *ICPADS '07: Proceedings of the 13th International Conference on Parallel and Distributed Systems - Volume 2 (ICPADS'07)*, pages 1–8. IEEE, 2007.

[22] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: the SimGrid simulation framework. In *Proceedings of the third International Symposium on Cluster Computing and the Grid*, pages 138–145. IEEE, 2003.

[23] I. C. Legrand and H. B. Newman. The MONARC toolset for simulating large network-distributed processing systems. In *Proceedings of the 32nd conference on Winter simulation*, pages 1794–1801. Society for Computer Simulation International, 2000.

[24] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1):46–61, 1973.

[25] X. Liu, H. Xia, and A. Chien. Validating and scaling the MicroGrid: A scientific instrument for Grid dynamics. *Journal of Grid Computing*, 2(2):141–161, 2004.

[26] P. Niemeyer and J. Knudsen. *Learning Java*. O'Reilly Media, third edition, 2005.

[27] M. Pinedo. *Planning and scheduling in manufacturing and services*. Springer, 2005.

[28] G. Ritchie and J. Levine. A fast, effective local search for scheduling independent jobs in heterogeneous computing environments. Technical report, Centre for Intelligent Systems and their Applications, University of Edinburgh, 2003.

[29] J. Skovira, W. Chan, H. Zhou, and D. A. Lifka. The EASY - LoadLeveler API Project. In *Job Scheduling Strategies for Parallel Processing*, volume 1162 of *LNCS*, pages 41–47. Springer, 1996.

[30] K.-U. Stucky, W. Jakob, A. Quinte, and W. Süß. Solving scheduling problems in Grid resource management using an evolutionary algorithm. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4276 of *LNCS*, pages 1252–1262. Springer, 2006.

[31] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya. A toolkit for modelling and simulating data Grids: an extension to GridSim. *Concurrency and Computation: Practice & Experience*, 20(13):1591–1609, 2008.

[32] A. Takefusa, S. Matsuoka, K. Aida, H. Nakada, and U. Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, page 11. IEEE, 1999.

[33] X. Tang and S. T. Chanson. Optimizing static job scheduling in a network of heterogeneous computers. In *ICPP '00: Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing*, pages 373 – 382. IEEE, 2000.

[34] F. Xhafa and A. Abraham. Meta-heuristics for grid scheduling problems. In *Metaheuristics for Scheduling in Distributed Computing Environments*, volume 146 of *Studies in Computational Intelligence*, pages 1–37. Springer, 2008.

[35] M. Q. Xu. Effective metacomputing using LSF multicluster. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, pages 100–105. IEEE, 2001.

[36] C. S. Yeo and R. Buyya. Integrated risk analysis for a commercial computing service in utility computing. *Journal of Grid Computing*, 7(1):1–24, 2009.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8722
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8722