Batsim: A Realistic Language-Independent Resources and Jobs Management Systems Simulator

Pierre-François Dutot[1,2], Michael Mercier[1,2,3], Millian Poquet[1,2(✉)], and Olivier Richard[1,2]

[1] Univ. Grenoble Alpes, LIG, 38000 Grenoble, France
[2] Inria, CNRS, LIG, 38000 Grenoble, France
{pierre-francois.dutot,michael.mercier,millian.poquet, olivier.richard}@inria.fr
[3] Atos, Bezons, France

Abstract. As large scale computation systems are growing to exascale, Resources and Jobs Management Systems (RJMS) need to evolve to manage this scale modification. However, their study is problematic since they are critical production systems, where experimenting is extremely costly due to downtime and energy costs. Meanwhile, many scheduling algorithms emerging from theoretical studies have not been transferred to production tools for lack of realistic experimental validation. To tackle these problems we propose Batsim, an extendable, language-independent and scalable RJMS simulator. It allows researchers and engineers to test and compare any scheduling algorithm, using a simple event-based communication interface, which allows different levels of realism. In this paper we show that Batsim's behaviour matches the one of the real RJMS OAR. Our evaluation process was made with reproducibility in mind and all the experiment material is freely available.

Keywords: RJMS · Scheduling · Simulation · Reproducibility

1 Introduction

Resources and Jobs Management Systems (RJMSs) play a critical role in modern high performance computing (HPC) infrastructures, simultaneously maximizing the efficiency of the platform and fairly sharing its capacities among all their users. Thus, the job scheduling algorithms that are used need to be effective in multiple domains. On the way to exascale computing, large scale systems become harder to study, to develop or to calibrate because of the costs in both time and energy of such processes. It is often impossible to convince managers to use a production cluster for several hours simply to test modifications in the RJMS. Moreover, as the existing RJMS production systems need to be highly reliable, each evolution requires several real scale test iterations. The consequence is that

N. Desai and W. Cirne (Eds.): JSSPP 2015/2016, LNCS 10353, pp. 178–197, 2017.
DOI: 10.1007/978-3-319-61756-5_10

scheduling algorithms used in production systems are mostly outdated and not customized correctly.

The most efficient way to tackle these problems is coarse-grained simulation. Simulation of these large scale systems is faster by multiple orders of magnitude than real experiments. The savings in computing time and energy consumption allow a much larger variety of scenarios to be tested. This unlocks new research avenues to explore, and possibly leads to scientific discoveries and industrial innovations.

Furthermore, recent algorithms developed in scheduling theory are impossible to compare in realistic conditions, because of the lack of simple and extensible simulators. There is a vast literature on possible theoretical improvements, proved in different theoretical job and platform models which are generally not yet transferred to production schedulers in real environments.

The research field around RJMS and scheduling in large scale systems in general would greatly benefit from a simple – yet realistic – validated scheduling simulator that is modular enough to be used with any theoretical or production algorithm implementation in order to truly compare all of them in a scientific way. Currently existing RJMS simulators are based on too simple models. Most of them only rely on delays for job modeling or on network models that are either minimalistic or not scalable enough to test really large scale systems.

From this assessment, we propose Batsim (for BATch scheduler SIMulator). It is based on SimGrid [7], a state-of-the-art distributed platform simulator with realistic network and computation models. Batsim allows different levels of realism depending on the user's needs, uses a simple message interface to achieve language independence, uses an easily expandable workload input format and provides readable and analysable outputs with jobs and scheduling details. For comprehension's sake, a simple Gantt chart visualisation tool is provided separately.

Batsim was also created to achieve experiment reproducibility. We are well aware that hardware suffers from a great variability. This is the main barrier to achieve experiment reproducibility in computer science. But in the case of simulation, those constraints do not exist anymore because the result of the simulation can be deterministic with respect to the simulation's inputs (parameters and input data). That is why simulation experiments are much easier to reproduce if the simulation environment and inputs are provided by the authors, which is rarely the case [29]. There can be several reasons for this, as explained in [31]:

- Restrictive licence or any intellectual property problem
- Complexity of the usually homemade simulation tool
- Missing experimental plan: used parameters are not provided
- Input data and/or results are not provided
- No access to the experimental environment (like the testbed or computer Grid)

Despite the intellectual property problem, which by definition prevents the reproducibility, all the aforementioned problems could be addressed by some good practice and appropriate tools:

1. Use reusable and proven simulators
2. Provide environments
3. Provide experiment plan, design and workflow
4. Provide inputs and results

Batsim was made to implement the first of these solutions. In fact, most published articles use ad hoc simulators which are not expected to be used after the articles' publications. Furthermore, simulators kept on-the-shelf are not proven to be accurate. In order to validate simulation results, the simulator must be assessed theoretically, and also experimentally if possible. Batsim aims at improving repeatability in this domain which is widely affected by the problems mentioned above.

The rest of this paper is organised as follows. Section 2 presents related work in the field of RJMS simulation. Section 3 gives an overview of how Batsim works. Section 4 develops the underlying models of Batsim. Section 5 gives more detailed explanations on Batsim's mechanics. Batsim's evaluation process is presented in Sect. 6, and its results in Sect. 7. Section 8 gives technical details on how to repeat our evaluation process. Section 9 concludes the paper and outlines our future work.

2 Related Work

Many simulators can be found in the literature which can be used to simulate a RJMS. Unfortunately, most implementations are either not available online or depend on outdated softwares and libraries which are themselves not available anymore. Thus, we chose to focus on simulators whose source code could be found.

To the best of our knowledge, most scheduling simulators are either very specific to one domain (e.g. Realtss) or do not primarily focus on realism of results, since comparisons to real existing systems are hardly ever done. This can be easily explained by the financial and ecological cost of such evaluations.

The approach which is closest to ours may be Alea [21]. This simulator is based on the GridSim simulation toolkit and allows to compare different scheduling algorithms. We chose the same approach of building our scheduling simulator on top of a simulation framework. However, Batsim's approach and Alea's differ in their modularity since Batsim allows to connect any scheduling algorithm, written in any programming language, whereas the supplied Alea API only allows new schedulers to be written in Java inside the project source code, with a queue-oriented API. Moreover, at the best of our knowledge, this simulator has not been validated in a real environment yet.

Another interesting approach can be found in article [26]. This approach consists in using the INSEE [27] fine-grained network simulator offline, in order

to obtain the execution time of any configuration the job can run in. Article [26] proposes job placement policies that guarantee that no network interference can occur between the jobs, allowing to use offline execution times while simulating an online workload. However, this approach cannot be used accurately when jobs can interfere with each other.

A previous initiative of building a scheduling simulator on top of SimGrid has been done in Simbatch [5]. However, as far as we know, the project has been left unmaintained since 2007 and cannot be used easily with current SimGrid versions. Moreover, Simbatch has not been developed in the perspective of connecting real RJMSs into it, nor to allow separation of concerns regarding system simulation and scheduling algorithms.

3 Batsim General Description

Batsim is an open source platform simulator that allows to simulate the behaviour of a computational platform on which a workload is executed according to the rules of a scheduling algorithm. In order to obtain sound simulation results and to broaden the scope of the experiments that can be done thanks to Batsim, we did not choose to build it from scratch but on top of the SimGrid simulation framework instead.

Batsim allows separation of concerns since it decouples the platform simulation and the decisions in two clearly separated components, represented in Fig. 1. The Batsim component is in charge of simulating the computational resources behaviour whereas the Decision component is in charge of taking scheduling or energy-related decisions. The scheduling decisions include executing a job or rejecting it. The energy-related decisions include changing the power state of a machine – i.e. to change its DVFS mode – or switching a machine ON or OFF.

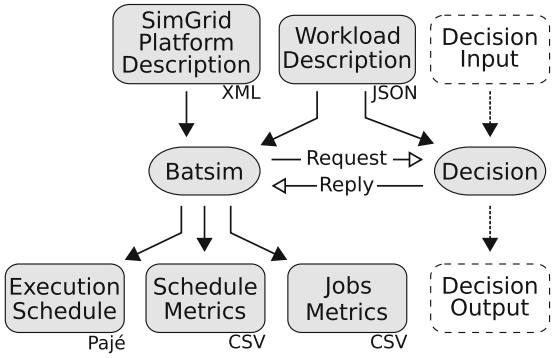

Fig. 1. The two *real* processes involved in a Batsim simulation, their network communication pattern and their inputs and outputs.

The components are instantiated as processes (within the meaning of Operating System processes) and communicate via a Unix Domain Socket.

The communication protocol used is a simple synchronous one which follows the request-reply pattern. The simulation runs in the Batsim component and as soon as an event occurs, Batsim stops the simulation and reports what happened to the Decision component. Batsim then waits for the Decision component's reply, then continues the simulation by applying the Decision component's choices. Protocol details can be found in [34].

Splitting the simulation workflow into two separated components in this way allows the Decision component to be implemented in any programming language which can communicate through a Unix Domain Socket. Thus, existing event-based scheduling algorithm implementations would not be hard to adapt in order to connect them with the Batsim component.

4 Models

This section describes the simulation models used by Batsim by giving a simplified overview of SimGrid in Sect. 4.1 then by detailing the models specific to Batsim in Sect. 4.2.

4.1 SimGrid Models

Since Batsim uses SimGrid internally, a brief – and simplified – overview of the SimGrid models that were used is given for sake of clarity. A **host** is a resource that can compute floating-point operations (flop). Hosts are connected via a network whose **links** have a latency (in seconds) and a bandwidth (in bytes per second). Links are hyperedges which can either represent a network link (a connection between two nodes) or a network node (a switch or a router). Hosts and links compose a platform that can be of virtually any topology.

Several SimGrid **processes** can be run on any SimGrid host. A SimGrid process can only be on one host at a time. These SimGrid processes – which will simply be called processes from now on – are user-given source code executed within the simulation. These processes can execute **tasks** and communicate with each other with **messages**. For simplicity's sake, we will assume that such messages are sent to processes.

Hence, a SimGrid-based simulator is composed by a set of processes which compute user-given functions. The user-given functions are executed within the simulator whereas the execution of tasks and the communications are simulated. Please note that user-given functions can spawn processes on which user-given functions are executed. SimGrid orchestrates, at any time, which processes are executed and which ones are not. SimGrid may change the running processes whenever they enter a state that implies a simulation time increase. For example, computing a task, sending a message or entering a sleep state implies a simulation time increase.

SimGrid allows us to create parallel tasks which combine computations and communications. To do so, we can build tasks with a computation vector (or row matrix) c where each c_k represents the amount of computation (in number of

floating-point operations) that must be computed on the k^{th} host on which the parallel task is run and a square communication matrix C in which each element $C[r, c]$ represents the amount of communication (in number of bytes) from the r^{th} to the c^{th} hosts involved in computing the task.

The SimGrid energy consumption model only takes computation hosts into account at the moment and is described in the following paragraph. Each host h has a set of power states P_h where each power state $p \in P_h$ has a computational power cp_p (in number of floating-point operations per second), a minimum electrical power consumption $ep_p^{\wedge}$ (in W) and a maximum electrical power consumption $ep_p^{\vee}$ (also in W). The SimGrid model which simulates the execution of activities on simulated resources computes the computational load $l_h(t)$ of each host h at any simulation time t. The load is a real number in [0, 1] where 0 represents an idle host and 1 a host computing at maximum computational power. Let $p_h(t)$ be the power state in which host t is at simulation time t. The instant electrical consumption of host h at time t is noted $P_h(t)$ and is determined by $p_h(t)$ and $l_h(t)$. $P_h(t)$ is computed as the linear interpolation between $ep^{\wedge}_{p_h(t)}$ and $ep^{\vee}_{p_h(t)}$ in function of $l_h(t) : P_h(t) = ep^{\wedge}_{p_h(t)} + (ep^{\vee}_{p_h(t)} - ep^{\wedge}_{p_h(t)}) \cdot l_h(t)$ and is expressed in watts. The energy consumption of host h is then given by $E_h = \int P_h(t)dt$ and is expressed in joules.

4.2 Batsim Models

The computation platforms used by Batsim are theoretically as broad in scope as SimGrid ones and can be of virtually any topology. However, only a subset of SimGrid platforms are valid Batsim platforms. Indeed, we chose to use a dedicated SimGrid host – that may be also be referred to as *computational resource* from now on – referred as the *Master host* to compute the resource management processes. In order to be able to run a job on computational resources, the platform must allow messages to be exchanged between the Master host and the other computational resources. Moreover, if jobs are parallel and must be run on different computational resources, the platform must allow the set of computational resources allocated to the job to communicate with each other.

Moreover, we enhanced the SimGrid energy model by adding explicit sleep and transition power states. We chose to split the set P_h of power states of the host h into three disjoint sets: P_h^c is the set of **computation** power states, P_h^s is the set of **sleep** power states and P_h^t is the set of **transition** power states. The computation power states are the only ones which can be used to compute jobs. A sleep power state represents the state of one machine which cannot instantaneously compute something *e.g.* ACPI S1, S3, S4 or S5 states. A Batsim host can switch from one computation power state to another instantaneously. However, entering into one sleep power state s or leaving it can take some time and cost some energy. Transition power states are *virtual* power states which are only used to simulate the transition into and from sleep power states. To do so, the amount of computation done in one transition is fixed to 1 flop. If one transition t should take time t_t (in seconds) and consume e_t energy (in joules), the corresponding virtual power state p_t should have a computational power

$cp_{p_t} = \frac{1}{t_t}$ and electrical power consumption bounds $ep^{\wedge}_{p_t} = ep^{\vee}_{p_t} = \frac{e_t}{t_t}$. If Batsim is run with energy enabled, the platform used must fulfill all SimGrid energy requirements and define, for each host h each sleep power state $s_h \in P^s_h$, the transition power state $v^{\downarrow}_{s_h}$ used to switch into s_h and the transition power state $v^{\uparrow}_{s_h}$ used to leave s_h.

Batsim workloads are divided into two parts: one set J of jobs and one set P of profiles. Each job $j \in J$ must have a unique job number id_j, a submission time sub_j (the time at which the job is submitted into the system), a walltime $wall_j$ (the user-specified execution time bound such that j is killed if its execution time exceeds $wall_j$), the number of requested resources res_j (rigid at the moment but moldable jobs can trivially be added into our architecture if needed) and the **profile** $prof_j$ the job executes. One profile describes how a job is computed. The profile information has been separated from the jobs because 1. it avoids data duplication when many jobs are computed in the same way 2. it makes workload generation easier and more modular.

At the moment several atomic profile types are available: **Delay** profiles are fixed amounts of time, **msg** profiles compute a vector c and a communication matrix C and **smpi** profiles simulate the execution of one SimGrid MPI time-independent trace within Batsim. Moreover, non-atomic profile types exist such as the **msg_homogeneous** profile type that wraps the msg profile type and simplifies its usage by forcing homogeneity to the underlying computation vector and communication matrix. As another non-atomic profile type, we can think of the **sequence** profile type, which is composed of a list of other profiles it must execute in sequence a certain number of times. The sequence profile type can be used to model Bulk Synchronous Parallel jobs for example. Our architecture allows to implement new profile types quite easily and the JSON format used in workload description allows modularity since any user can add any field to jobs or profiles to match their needs. For example, we used the same workloads to compare Batsim's behaviour to a real platform's by simply specifying how each profile should be run on the real platform.

5 Batsim Inner Mechanics

Batsim is a C++ program developed using the SimGrid C library. SimGrid allows us to simulate the behaviour of a computation platform on which concurrent SimGrid processes are run. We will use the phrase "real process" to talk about a process which is directly run by the user on a given Operating System. For example, the Batsim real process and the Decision real process. A SimGrid process (which will simply be referred to as a *process* from now on) is a process simulated within Batsim. The main Batsim processes and their interactions are shown in Fig. 2.

The **Jobs Submitter** process is in charge of reading one workload and submitting the jobs at the right simulation time. To do so, it simply iterates over the jobs in ascending submission time order and sends a message to the Server process as soon as a job has been submitted. If the next job submission time is

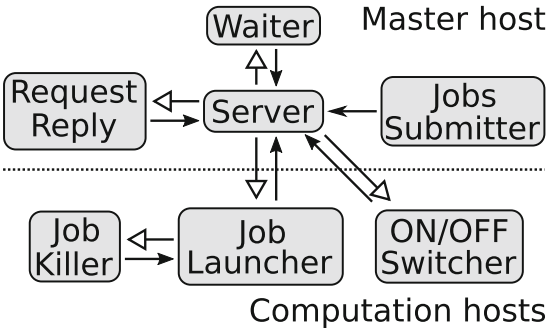

Fig. 2. The different processes within Batsim. The Job Launcher, Job Killer, and ON/OFF Switcher processes are executed on the computational resources whereas the other processes are run on a dedicated resource called the Master host. Filled arrows represent message transmissions and hollow arrows stand for a process execution.

strictly after the current simulation time, this process enters a sleep phase until the aforementioned job is submitted. Once all the jobs of its appointed workload have been submitted, the Jobs Submitter process tells the Server it has finished then ends. Several Jobs Submitters may be instantiated in the same simulation, which can be useful to simulate internally (within Batsim) the behaviour of concurrent users. These processes are executed on the Master host.

The **Request Reply** process's role is to manage the Unix Domain Socket to communicate with the Decision real process. Its lifespan is short: It is launched by the Server process on the Master host, it sends a network message to the Decision real process, then waits for its reply. It then parses the received message according to the Batsim protocol. The received message is composed of a sequence of events. In order to simulate the impact of real decision algorithms, which may take some decisions before returning the control flow, a simulation time is associated to each event (potentially the same for all events). The Request Reply process parses and transmits those events to the Server process one by one in the given chronological order. Just as the Jobs Submitter process, the Request Reply process enters a sleep phase between events to respect the received simulation times at which the events should have occured. Once all events have been sent to the Server, the process tells the Server the message has been managed, then ends.

The **Server** is the main Batsim process. It orchestrates the simulation: It knows when jobs are submitted, chooses when the Decision real process should be requested – ensuring that only one Request Reply process can be executed at the same time – and follows the orders dictated by the Decision real process (received from the Request Reply process). Depending on the received order, the Server process might launch a Job Launcher process to run a job, or launch a ON/OFF Switcher process to either boot or shutdown a computational resource, or launch a Waiter process if the Decision real process wants to be awaken at a specific time. The Server process ends its execution if and only if the following

conditions are met. 1. All Jobs Submitters must have finished their executions. 2. All submitted jobs must have been either processed or rejected. 3. The Request-Reply process must not be running at the current time. 4. All ON/OFF Switchers must have finished their executions. 5. There should be at least one Jobs Submitter connected – to ensure that the simulation starts.

The **Waiter** process is used to wake the Decision real process at a specific time. To do so, it is launched by the Server on the Master host, sleeps for the right amount of time, tells the Server it is time to wake the Decision real process then ends.

The **ON/OFF Switcher** process is in charge of simulating what happens when a computational resource is booted or shut down. It is executed on the computational resource whose state is requested to change.

The **Job Launcher** process is in charge of executing a job on a set of computational resources R. The process is executed on one computational resource $r \in R$. Jobs are then computed according to their profiles. If the profile type is Delay, the process is simply put into a sleep state for the right amount of time. If the profile type is MSG, the computation vector and the communication matrix are generated into memory then computed on the given set of computational resources R. If the profile type is Sequence, the different subjobs are computed sequentially according to previously given rules for the requested number of times. Finally, if the profile type is SMPI, the given MPI trace is replayed on R. To respect the walltime of non-SMPI jobs, the Job Launcher process executes the **Job Killer** process. The Job Killer process is in charge of waiting for the walltime amount of time. For each walltime-constrained job j there is a double $p_j = (launch_j, kill_j)$ where $launch_j$ is the Job Launcher process in charge of job j and $kill_j$ is the Job Killer process associated to $launch_j$. The first process to finish its task in p_j (either the job computation or the sleep) cancels the other process's task. This leads to $kill_j$ finishing its execution and $launch_j$ sending a message to the Server process to tell it j has been computed (successfully or not) then finishing its execution too.

6 Batsim Evaluation Experiment

In order to evaluate whether Batsim's behaviour is similar to real RJMSs', we set up an experiment comparing Batsim to OAR [6]. OAR is a RJMS – or batch scheduler – notably known for being used in the Grid'5000 [1] infrastructure. We chose OAR over other RJMSs – e.g. SLURM – because the modular design of OAR allows its scheduling part to be decoupled from the other parts of the system very easily. We tested Kamelot [35], a conservative backfilling scheduling algorithm implemented in OAR by executing it on real OAR-managed resources on Grid'5000 and by plugging it to Batsim. Since the same scheduling algorithm is used both in reality and in simulation, this allows us 1. to demonstrate that Batsim's architecture can be used to test production schedulers 2. to check Batsim's behaviour's soudness.

Our experimental process can be simplified in two major parts: 1. workload generation 2. schedule comparison between real and simulated workload executions. The first major part requires real programs, their instrumentation and a methodology to create jobs in our different job models. This is detailed in Sect. 6.1. Subsequently, the way the jobs are put together to form workloads is explained in Sect. 6.2. The second major part is described in Sects. 6.3 and 6.4.

Using a realistic simulator means using a realistic platform. Fortunately, the Graphene cluster situated in the Nancy Grid'5000 site [13] has already been calibrated so we chose to use this cluster for our real and simulated experiments. Thus, all our real experiments were done on the Grid'5000 Graphene cluster, reserving the nodes below one switch each time.

6.1 Profile Generation

In order to execute a scheduling algorithm, workloads must be generated. However, the jobs of our workloads must fulfill some requirements to be executed both in a real platform and in simulation. Batsim allows different levels of realism depending on the profile models used in the workload, which makes the workload generation process more complex. Indeed, the msg model needs realistic computation vectors and communication matrices to make sense. Furthermore, the smpi model requires MPI traces in order to be used.

In order to obtain realistic values for our profile models, we chose to execute real jobs from the MPI version of the NAS Parallel Benchmarks (NPB)[1] and to instrument them to obtain execution traces. We have selected the three benchmarks IS, FT and LU. We chose to compile and execute them for all available processor sizes – powers of two from 1 to 32 – and for tiny to medium data sizes – B to D depending on the benchmark. Considering NPB limitations, we were able to compile 47 different MPI programs.

First, in order to obtain the real execution times of our programs, we run them in a sequence (one by one, to avoid network influence of one program to another) without instrumentation. This allowed us to directly generate delay profiles.

We then instrumented the jobs using Extrae [2], which gave us heavy execution traces. In order to get time-independent traces – required by SimGrid – from the format used by Extrae, we used a script, courtesy of Lucas Schnorr [33]. Time-independent traces contain, for each processor, a sequence of events describing how many flops the processor computed or the MPI functions it called with the associated data amount. Unfortunately, the conversion script we used is a work in progress and was not able to capture all MPI messages, which added a profile calibration phase in our experiment process.

Since SimGrid does not allow – at the moment – the concurrent execution of several SMPI applications at the same time, we were not able to validate the smpi profile type in the present article. On the other hand, we aggregated the time-independent traces into computation vectors and communication matrices

[1] NPB 3.3.1 available here [25].

to obtain msg profiles. These msg profiles have been calibrated such that their execution times match those of the previously obtained delay profiles. The main advantage of the calibrated msg profiles over the delay ones is that their execution time may vary depending on resources' computational power, network bandwidth or network contention. On the contrary, delay profiles have a fixed execution time that strongly depends on the platform they were executed on, and cannot take network contention into account. This difference makes this type of profile more appropriate for heterogeneous experiments.

6.2 Workload Generation

The workload generation algorithm that we decided to use is described in this paragraph and was inspired by Chap. 9.6 of book [12]. Please note that our workload generation method is not intended to be sound for comparing scheduling heuristics, but only to evaluate how Batsim behaves compared to a real RJMS. The algorithm generates $N = 800$ jobs iteratively. The interarrival submission times of the jobs is computed randomly with a Weibull distribution of shape parameter $k = 2$ and scale parameter $\lambda = 15$. Since the job sizes (the rigid number of resources a job requests) of the real jobs at our disposal are powers of 2 (from 1 to 32), the size of each job is computed with the formula $2^{\lfloor u \rfloor}$ where u is a lognormal variate of parameters $\mu = 0.25$ and $\sigma = 0.5$. Only variates such that $\lfloor u \rfloor$ is in $[0, log_2(32) = 5]$ are used to match the sizes at our disposal. The generation of those workloads depends on a random seed, simply referred as *seed* in the remainder of this article.

We chose to generate nine different workloads and to execute each of them once below two different switches of the Graphene cluster. We chose two different switches to obtain more representative simulation results. Indeed, both the computation nodes and the switches are homogeneous in Graphene and are described in the exact same way in the simulated platform. However, in practice, little differences exist between nodes that are supposed to be identical and we hope that these differences will be more noticeable this way.

6.3 Executing Workloads in a Real Platform

In order to execute the workloads we generated on Graphene, we used a reproducible methodology which is described in Sect. 8. This methodology includes the installation and the configuration of OAR within the nodes we reserved in Graphene. We configured OAR such that it uses the Kamelot scheduler and implemented a replay tool that reads a Batsim workload, then launches real OAR submissions at the times dictated by the workload. The OAR submissions launch the MPI programs which were previously generated.

6.4 Executing Workloads in Simulation

Executing the workloads in simulation simply consists in running the Kamelot scheduler on Batsim with the aforementioned calibrated platform file. To do so, we created an adaptor between Batsim and OAR which will be used later to test the different algorithms implemented in OAR.

7 Results

The nine different workloads we generated have been executed twice on a real platform (on identical machines and network, but not on the exact same machines), and twice in simulation (with Delay and MSG profile types). This section presents the different results and analyses them.

An overview of the execution of all the workloads can be found on Fig. 3. First of all, Fig. 3 shows that the MSG and the Delay simulation results are very close to each other. The execution times of MSG jobs depends on where the jobs are allocated and depends on the network saturation. However, in this experiment, the platform is highly homogeneous and very low contention has been observed during the jobs' execution, which explains why the results are so close. Furthermore, Fig. 3 shows that the difference between two real executions of the same workload is not negligible.

7.1 Similarities

Many similarities exist between the schedules resulting from the simulated and real executions of the workloads we defined.

Figure 3 allows to see the makespan and the mean bounded stretch of real and simulated executions of all the workloads we generated. The closer the points of a given color, the more similar the real and simulated executions are. For most

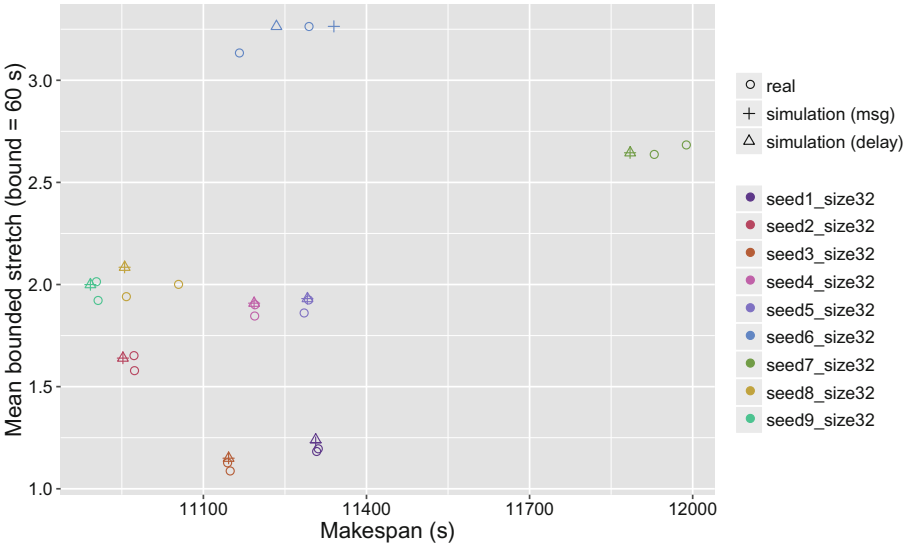

Fig. 3. Mean bounded stretch against the makespan of all workload executions. Each point represents one workload execution. Circles are executions on the real platform, triangles and crosses are simulated executions with respectively Delay and MSG profiles. Each workload is associated with one color.

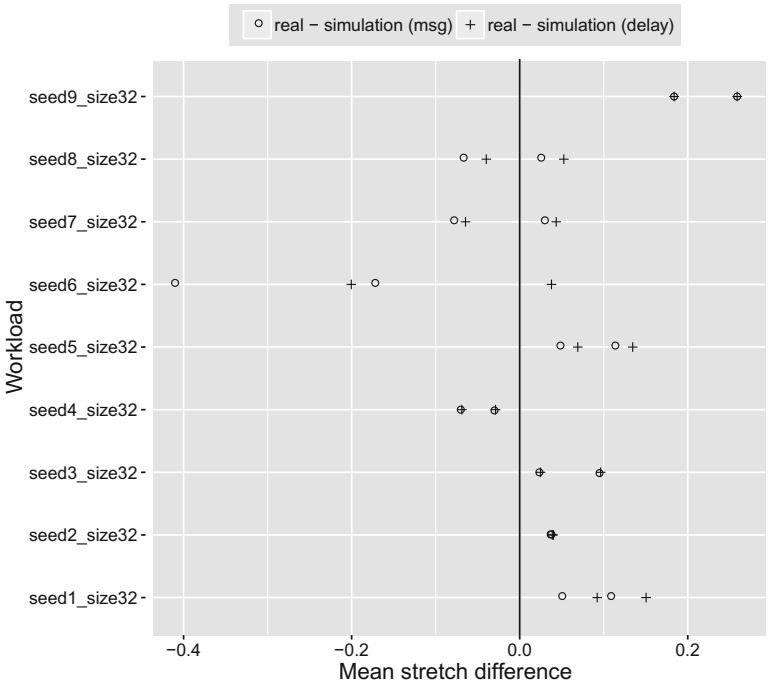

Fig. 4. Mean stretch difference between real and simulated executions of each workload. Different workloads are plotted on different lines. Circles and crosses represent the difference from the simulated execution with respectively MSG and Delay profiles.

workloads, the points are tightly clustered. Workloads 6, 7 and 8 seem to be unstable in both makespan and mean bounded stretch when they are executed with this scheduling algorithm. For these three workloads, we can see that the points are less tightly clustered. Furthermore, the distance between the points resulting from real and simulated execution is of the same order of magnitude as the one resulting from two real experiments.

Figures 4 and 5 shows the differences in mean stretch between real and simulated executions of all the workloads. The mean stretch of real different executions is in range $[1.492, 5.876]$. Figure 5 shows that the mean stretch difference is centered a little bit after zero if we look at all the workloads at once. Figure 4 shows that the mean stretch of the simulated execution of each workload is not necessarily between the two values coming from real executions, but that the mean stretch difference between the real and the simulated executions of one workload is of the same order of magnitude as the mean stretch difference between the two real executions of the same workload.

7.2 Differences

Figure 6 shows that simulated workload executions are almost always below real executions, which means that Batsim underestimates the waiting time of jobs.

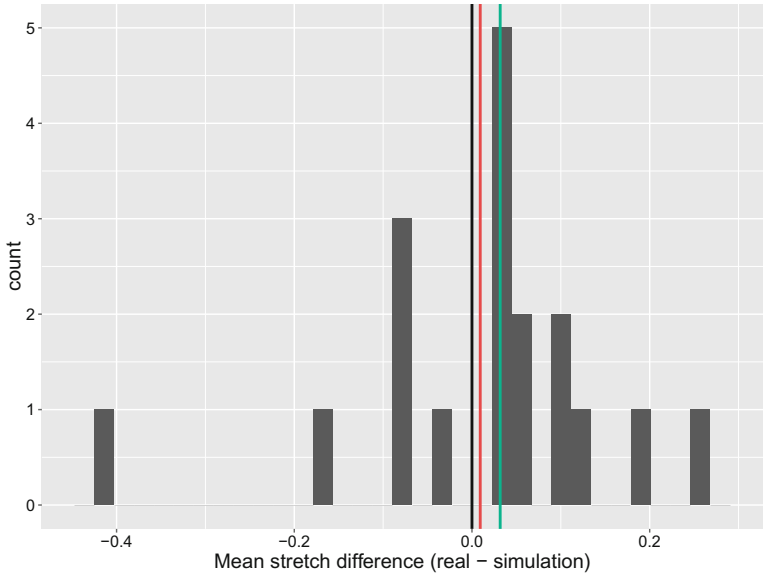

Fig. 5. Distribution of the mean stretch difference between real and simulated (with MSG profiles) executions of all workloads. The black vertical line is on zero. Red and green vertical lines are respectively the mean and median of the mean stretch difference among all workloads. (Color figure online)

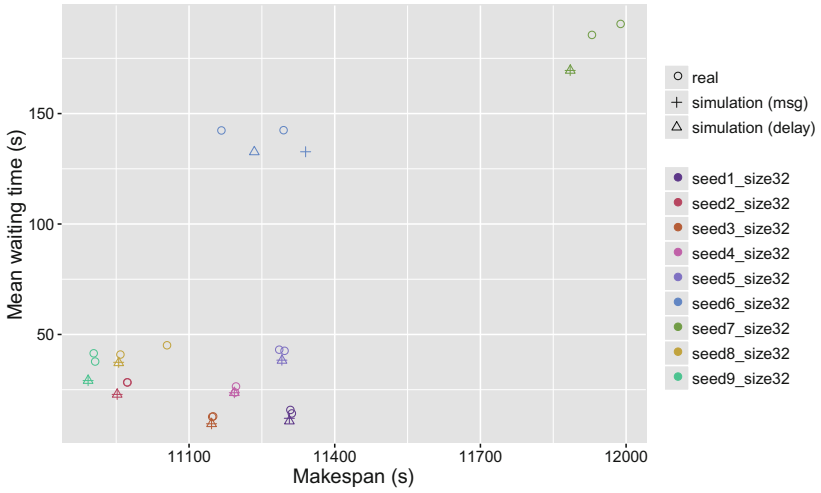

Fig. 6. Mean waiting time against the makespan of all workload executions. Each point represents one workload execution. Circles are executions on the real platform, triangles and crosses are simulated executions with respectively Delay and MSG profiles. Each workload is associated with one color.

This underestimation can be explained by the OAR's ssh-based job launching procedure and the OAR's job cleaning procedure. Indeed, these procedures take a non-negligible amount of time and have not been modeled into Batsim. The way the jobs are launched and cleaned is highly RJMS-dependent and we chose not to overfit any RJMS for the moment.

Furthermore, differences can be observed if we look at each workload at a finer grain. For example, the Gantt charts of the real and the simulated executions of the same workload differ, but this is also true for two real executions of the same workload, as seen in Fig. 7.

8 Reproducing Our Work

One of our main goal while creating Batsim was to foster reproducibility in the field of jobs and resources scheduling, by providing the tools needed to make more reproducible science. The aim of this section is to explain how all our evaluation process – or only a part of it – can be reproduced. To do so, we provide a complete environment to use Batsim and the different schedulers which run on top of it.

All the experiment tools mentioned below (Batsim, Kameleon, Execo, Grid'5000) are necessary to repeat the experiments we done. Of course, other tools exist to achieve reproducibility but we describe the ones we chose for our experiments.

Environments. An environment can be seen as an object that fixes a software context. Such an environment typically regroups an Operating System and a set of programs and libraries, specifying which version is used for each component. To build our environments we used Kameleon [30]. This tool allows its users to build environments from template recipes and to extend them. It also allows a user to write their own environment recipes from scratch. Such environments handle failures thanks to a breakpoint mechanism. Recipes can be shared using Git, and Kameleon comes with the possibility to rebuild the environment from cache to achieve full re-constructibility. The software appliance produced by Kameleon can be exported to many formats, including a virtual machine, a docker container or a tarball, in order to be deployed anywhere.

The Batsim complete environment, as the workload generation environment recipes, are both available in the Git repository [17].

Experiment Design and Workflow. Most of the time the experiment design consists in one or more documents that describe the purpose and the experiment with some details and some dedicated scripts. Some domain specific tools exist to compute the experiment on a grid from a user-defined workflow [37], but it is not well suited for computer science experiments, which also need to select the underlying software stack and OS. Hopefully, computer scientists dedicated testbeds exist, like Grid'5000 which allows this level of management.

Batsim's evaluation experiment has been made using Execo [19], a tool which completely automates the experiment workflow. Execo is a tool which allows

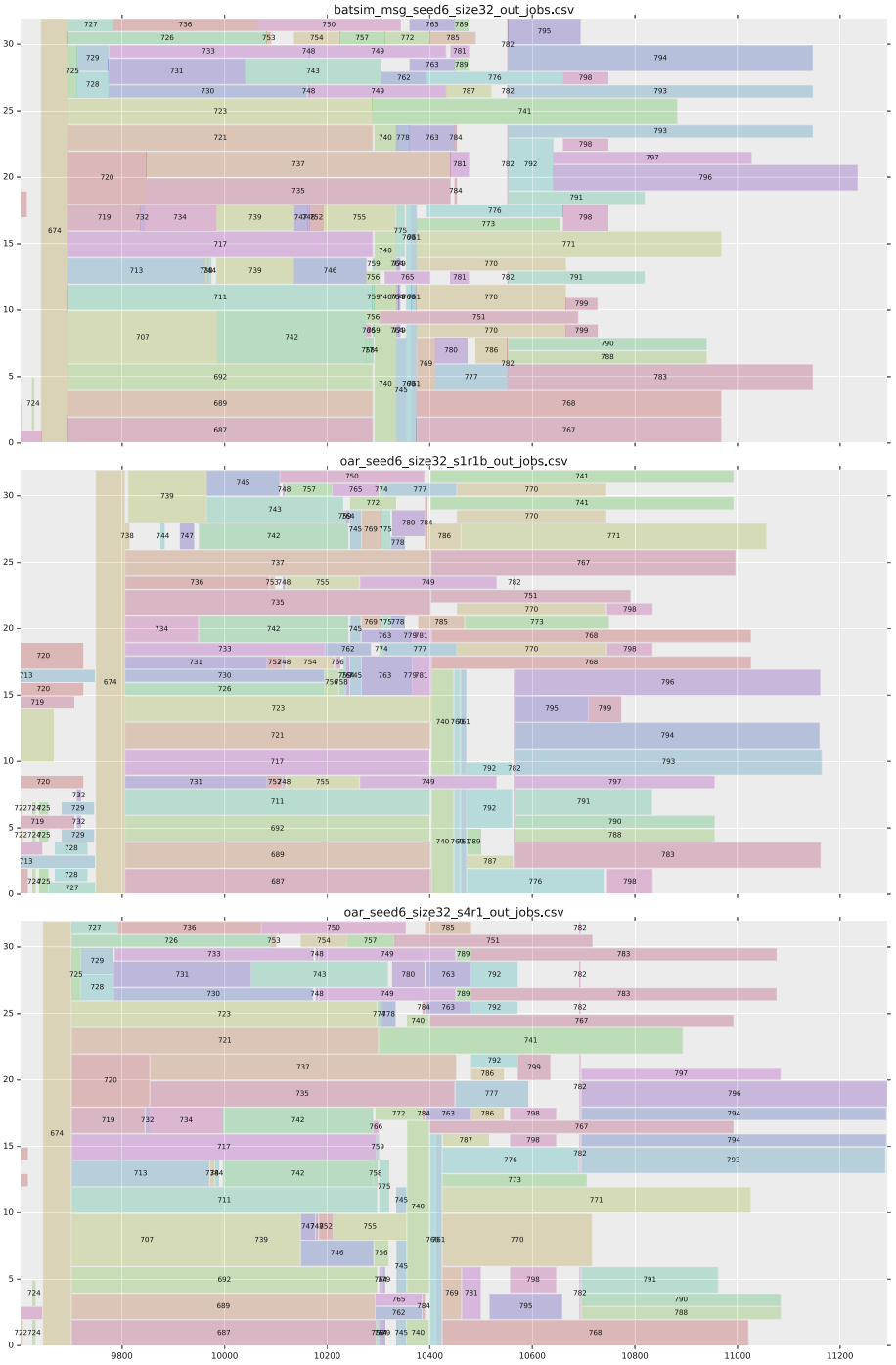

Fig. 7. Final section of the Gantt charts of the executions of workload $seed = 6$. The uppermost gantt chart is a simulated execution while the other two are real executions. Workload $seed = 6$ is the least stable workload in makespan.

Grid'5000 users to programmatically describe their experiment workflows in order to compute them on the grid. It is a Python toolbox library that allows to run local and remote processes easily. It also provides an engine to manage the parameters sweeping and an interface to the Grid'5000 testbed, which allows to design fully automated sets of experiments.

Moreover, the scripts and tools used to generate all the figures present in this paper are provided in our Git repository [18]. The Gantt chart visualisation and comparison tools named *Evalys* is available independently [15].

The complete experiment workflow made to conduct this paper's experiment is also available in our Git repository [18].

Inputs and Results. The original input data are crucial in the process of reproducibility. Most of the inputs and results of the experiments we have done are available in the aforementioned Git repository. The results that do not fit in the repository because of their size – notably the MPI instrumentation traces ($\simeq$20Go) – are shared using Academic Torrent [24].

9 Conclusion and Future Work

In this paper we presented Batsim, a modular RJMS simulator that provides different levels of realism, on which scheduling algorithms can be easily connected and compared. Batsim JSON-formatted workloads are extensible and allow painless workload generation. Batsim CSV outputs provide clear information about scheduling metrics, job placement and energy consumption, and can be easily linked with standard data analysis tools.

We used the OAR RJMS to check whether Batsim's behaviour is close to real RJMSs' in our experiment. In this experiment, the execution times of the jobs in the delay and msg profile models were almost identical because no contention has been observed during the experiment, and because the platform that we used was completely homogeneous both in computing power and in network bandwidth. As a future work, we can think of a validating process concerning the msg profile type, which may focus on conducting experiments on real heterogeneous platforms for example. Furthermore, the Batsim energy model has not been validated in real machines yet, which opens a door for future work.

We are well aware that the workloads used in our evaluation process remain small in their number of resources, their number of different jobs and in their duration. We would like to do larger scale experiments but finding funding to conduct this kind of study becomes problematic, as the energy and financial costs of reservations for such experiments would skyrocket.

We chose not to overfit the behaviour of a given RJMS, which may impact result realism on different metrics such as the mean waiting time, as seen in Sect. 7.2. Since our architecture allows to model finely the different RJMS's procedures, it would be beneficial to allow Batsim's users to parametrize how the different procedures should be done in order to improve accuracy.

At the moment, Batsim allows a production scheduler to be used in simulation. To do so, Batsim is in charge of simulating the RJMS whereas the

scheduling component of the real RJMS makes the decisions. An interesting future work would be to allow the opposite type of connection: The real RJMS would run normally but it would communicate with a Batsim-compatible algorithm to make scheduling decisions. This would simplify greatly the production launch of theoretical scheduling algorithms.

Even if Batsim is fully operational as we wrote this article, we would like to improve its capabilities in several way. For example, we would like to implement the possibility to concurrently run several SMPI application within SimGrid in order to use those applications within Batsim. Moreover, we are also interested in IO-related problems for big data workload simulation. Eventually, we want to implement a user model to take user reactions into account during the ongoing scheduling execution.

As we want to promote experiment reproducibility, all the materials necessary to understand and reproduce our evaluation experiments are provided online.

Batsim is an open source [16] project and we encourage any researcher or engineer that has to deal with resources and jobs scheduling to use it. Besides, we would be pleased to collaborate with anyone who wants to port an existing scheduling algorithm to Batsim. This would enhance the list of supported algorithms which may be studied and compared in the same context, in order to make more reproducible and better science.

References

1. Balouek, D., et al.: Adding virtualization capabilities to the grid'5000 testbed. In: Ivanov, I.I., Sinderen, M., Leymann, F., Shan, T. (eds.) CLOSER 2012. CCIS, vol. 367, pp. 3–20. Springer, Cham (2013). doi:10.1007/978-3-319-04519-1_1
2. Barcelona Supercomputing Center: Extrae, February 2016. https://www.bsc.es/computer-sciences/extrae
3. Bedaride, P., Degomme, A., Genaud, S., Legrand, A., Markomanolis, G., Quinson, M., Stillwell, M., Suter, F., Videau, B.: Toward better simulation of MPI applications on ethernet/TCP networks, November 2013. https://hal.inria.fr/hal-00919507/document
4. Bell, W.H., Cameron, D.G., Millar, A.P., Capozza, L., Stockinger, K., Zini, F.: Optorsim: a grid simulator for studying dynamic data replication strategies. Int. J. High Perform. Comput. Appl. **17**(4), 403–416 (2003)
5. Caniou, Y., Gay, J.-S.: Simbatch: an API for simulating and predicting the performance of parallel resources managed by batch systems. In: César, E., et al. (eds.) Euro-Par 2008. LNCS, vol. 5415, pp. 223–234. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00955-6_27
6. Capit, N., Da Costa, G., Georgiou, Y., Huard, G., Martin, C., Mounié, G., Neyron, P., Richard, O.: A batch scheduler with high level components. In: IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005, vol. 2, pp. 776–783. IEEE (2005)
7. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. J. Parallel Distrib. Comput. **74**(10), 2899–2917 (2014). http://hal.inria.fr/hal-01017319

8. Clauss, P.N., Stillwell, M., Genaud, S., Suter, F., Casanova, H., Quinson, M.: Single node on-line simulation of MPI applications with SMPI, May 2011. https://hal.inria.fr/inria-00527150/document
9. Diaz, A., Batista, R., Castro, O.: Realtss: a real-time scheduling simulator. In: 4th International Conference on Electrical and Electronics Engineering, 2007. ICEEE 2007, pp. 165–168. IEEE (2007)
10. Dutot, P.-F., Poquet, M., Trystram, D.: Communication models insights meet simulations. In: Hunold, S., et al. (eds.) Euro-Par 2015. LNCS, vol. 9523, pp. 258–269. Springer, Cham (2015). doi:10.1007/978-3-319-27308-2_22
11. Estrada, T., Flores, D., Taufer, M., Teller, P.J., Kerstens, A., Anderson, D.P., et al.: The effectiveness of threshold-based scheduling policies in BOINC projects. In: Second IEEE International Conference on e-Science and Grid Computing, 2006. e-Science 2006, p. 88. IEEE (2006)
12. Feitelson, D.G.: Workload Modeling for Computer Systems Performance Evaluation. Cambridge University Press, Cambridge (2015). https://cds.cern.ch/record/2005898
13. Grid5000: Nancy: Home - Grid5000, February 2016. https://www.grid5000.fr/mediawiki/index.php/Nancy:Home
14. Imbert, M., Pouilloux, L., Rouzaud-Cornabas, J., Lébre, A., Hirofuchi, T.: Using the EXECO toolbox to perform automatic and reproducible cloud experiments, December 2013. https://hal.inria.fr/hal-00861886
15. Inria: InriaForge: Evalys: Projet Home. https://gforge.inria.fr/projects/evalys
16. Inria: BatSim Homepage, February 2016. http://batsim.gforge.inria.fr/
17. Inria: InriaForge: Batsimctn: Project Home, February 2016. https://gforge.inria.fr/projects/simctn/
18. Inria: InriaForge:expe_batsim: Project Home, February 2016. https://gforge.inria.fr/projects/expe-batsim
19. Inria: Welcome to execo–execo v2.5.3, February 2016. http://execo.gforge.inria.fr/doc/latest-stable/
20. Jones, W.M., Ligon III, W.B., Pang, L.W., Stanzione, D.: Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. J. Supercomput. **34**(2), 135–163 (2005)
21. Klusáček, D., Rudová, H.: Alea 2: job scheduling simulator. In: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, p. 61. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2010)
22. Legrand, A.: Simgrid Usages, January 2016. http://simgrid.gforge.inria.fr/Usages.php
23. Lucarelli, G., Mendonca, F., Trystram, D., Wagner, F.: Contiguity and locality in backfilling scheduling. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 586–595. IEEE (2015)
24. Mercier, M.: MPI+PRV+TIT-traces_nas-Benchmarks_2016-02-08-10-10-44, February 2016. http://academictorrents.com/details/53b46a4ff43a8ae91f674b26c65c5cc6187f4f8e
25. NASA: NAS Parallel Benchmarks, February 2016. https://www.nas.nasa.gov/publications/npb.html
26. Pascual, J.A., Miguel-Alonso, J., Lozano, J.A.: Locality-aware policies to improve job scheduling on 3D tori. J. Supercomput. **71**(3), 966–994 (2015)

27. Ridruejo Perez, F.J., Miguel-Alonso, J.: INSEE: an interconnection network simulation and evaluation environment. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 1014–1023. Springer, Heidelberg (2005). doi:10.1007/11549468_111
28. Phatanapherom, S., Uthayopas, P., Kachitvichyanukul, V.: Dynamic scheduling II: fast simulation model for grid scheduling using HyperSim. In: Proceedings of the 35th Conference on Winter Simulation: Driving Innovation, pp. 1494–1500. Winter Simulation Conference (2003)
29. Proebsting, T., Warren, A.M.: Repeatability and benefaction in computer systems research. Technical report, The university of Arizona (2015). http://reproducibility.cs.arizona.edu/v2/RepeatabilityTR.pdf
30. Ruiz, C., Harrache, S., Mercier, M., Richard, O.: Reconstructable software appliances with kameleon. SIGOPS Oper. Syst. Rev. **49**(1), 80–89 (2015)
31. Stanisic, L., Legrand, A.: Effective reproducible research with org-mode and Git. In: Lopes, L., et al. (eds.) Euro-Par 2014. LNCS, vol. 8805, pp. 475–486. Springer, Cham (2014). doi:10.1007/978-3-319-14325-5_41
32. Takefusa, A., Matsuoka, S., Nakada, H., Aida, K., Nagashima, U.: Overview of a performance evaluation system for global computing scheduling algorithms. In: Proceedings of the Eighth International Symposium on High Performance Distributed Computing, pp. 97–104. IEEE (1999)
33. tcbozzetti: tcbozzetti/trabalhoconclusao, February 2016. https://github.com/tcbozzetti/trabalhoconclusao
34. oar team: Batsim protocol description (2016), https://github.com/oar-team/batsim/blob/master/doc/proto_description.md
35. oar team: Kamelot (2016). https://github.com/oar-team/oar3/blob/master/oar/kao/kamelot.py
36. Xia, H., Dail, H., Casanova, H., Chien, A.: The microgrid: using emulation to predict application performance in diverse grid network environments. In: Proceedings of the Workshop on Challenges of Large Applications in Distributed Environments (2004)
37. Yu, J., Buyya, R.: A taxonomy of scientific workflow systems for grid computing. SIGMOD Rec. **34**(3), 44–49 (2005)