SCORE: Simulator for cloud optimization of resources and energy consumption

Damián Fernández-Cerero [a], Alejandro Fernández-Montes [a], Agnieszka Jakóbik [b], Joanna Kołodziej [c], Miguel Toro [a]

[a] *Escuela Técnica Superior de Ingeniería Informática, Universidad de Sevilla, Av. Reina Mercedes s/n, Sevilla, Sevilla 41012, Spain*
[b] *Institute of Computer Science, Cracow University of Technology, Warszawska 24, Cracow 31-155, Poland*
[c] *Research and Academic Computer Network (NASK), Kolska 12, Warsaw 01-045, Poland*

Keywords:
Data center
Energy saving
Cloud computing
Energy-aware scheduling

A B S T R A C T

Achieving efficiency both in terms of resource utilisation and energy consumption is a complex challenge, especially in large-scale wide-purpose data centers that serve cloud-computing services. Simulation presents an appropriate solution for the development and testing of strategies that aim to improve efficiency problems before their applications in production environments. Various cloud simulators have been proposed to cover different aspects of the operation environment of cloud-computing systems. In this paper, we define the SCORE tool, which is dedicated to the simulation of energy-efficient monolithic and parallel-scheduling models and for the execution of heterogeneous, realistic and synthetic workloads. The simulator has been evaluated through empirical tests. The results of the experiments confirm that SCORE is a performant and reliable tool for testing energy-efficiency, security, and scheduling strategies in cloud-computing environments.

1. Introduction

Cloud-computing (CC) and large-scale web services have had a notable impact in the data center scenario and the big-data environment, since they enable huge amounts of data to be processed in a reliable and distributed way. However, the CC services in the big-data era should meet new requirements from the end users, such as fast-response and low latency.

Major CC service providers, such as Google, Microsoft and Amazon, are constantly developing new applications and services. As the number of these services grows, many types of applications have to be deployed on the same hardware. Virtualisation of the resources enabled the resource utilisation to be improved by designing general and wide-purpose data centers. These facilities can handle an enormous workload with various requirements. Therefore, many well-known solutions, such as fragmenting the data center into a set of clusters that are responsible for executing only one kind of application, are no longer needed.

Large-scale data centers which execute heterogeneous workloads on shared hardware resources bring new challenges in addition to those inherent to small and medium-sized clusters, not least because scheduling such an amount of work may exceed the capacity of a centralized monolithic scheduler. In order to overcome this limitation, several scheduling models with different degrees of parallelism have been developed, such as the two-level approach of Mesos [1], the shared-state approach of Omega [2], and various approaches for scheduling in large-scale grid systems [3].

The aforementioned infrastructures usually consume as much energy as do many factories and small cities, and account for approximately 1.5% of global energy consumption [4].

Although, data centers may be used by users worldwide, they are usually deployed on a continental basis. Therefore, these facilities are usually under higher pressure during day-time hours than in night-time hours. This and other reasons, such as the fear of any change that could break operational requirements [5], and the complexity of the systems involved, lead to an over-provision of data-center infrastructures. This decision leads to servers being kept underused or in an idle state, which is highly inefficient from an energy-consumption perspective.

Many models have been implemented in order to minimise the energy consumption in data centers, such as chiller-free cooling systems, and hardware model improvements, such as dynamic voltage and frequency scaling (DVFS). However, another approach that may lower the energy consumption considerably in data centers is seldom implemented. This strategy involves switching off idle servers.

The strategies that shut down idle machines may have a negative impact in terms of performance of the whole system. The inactive machines would not be able to execute large workloads in a short time. Therefore, the optimal energy-aware strategies must guarantee an appropriate level of reduction of energy consumption. In order to test these strategies and to measure the impact in terms of performance and energy consumption, a trustworthy simulation tool is required. In addition, the chosen simulator has to be able to reproduce the conditions present in real data centers. This requirement is critical, since the simulation is usually the step prior to implementing these strategies in working data centers. Therefore, the "optimal" energy-aware cloud simulator should guarantee the following achievements: a) Low resource consumption – data centers are composed of thousands of data servers which consume a huge amount of energy; b) Simulation of the parallel-scheduling models – the monolithic-scheduling model may prove ineffective in the large-scale CC systems; c) Easy codes for replication and extensions –generic model focused on shut-down, power-on and scheduling strategies, since many low-level aspects, such as hardware and networking details, may be overlooked, and d) Validation of results against a trustworthy source. This means that the simulation tool has been compared with the real-life system that it simulates.

Several simulators were evaluated in this paper. These tools followed various simulation models, such as: a) Discrete-event systems; b) Multi-agent systems; c) Multi-paradigm systems; and d) Hierarchic systems. The simulators under evaluation presented a wide range of purposes and levels of detail. However, by using these simulators, it is very difficult to achieve the aforementioned properties. In this paper, we propose a new data-center simulation tool, namely the *SCORE Cloud simulator*, which is our proposal for an "optimal" energy-aware CC simulation package. SCORE is based on the Google Omega lightweight simulator [2], which was extended by the implementation of the hybridisation of the discrete-event and the multi-agent scheduling and resource utilisation models. The Google Omega lightweight simulator has been validated against Google data centers, which makes it suitable for being the core of a trustworthy simulation tool. This is critical, since the authors have not been able to validate SCORE against real-world data centers due to the huge size of the clusters taken into consideration.

The paper is organised as follows. In Section 2, a simple comparative analysis is presented of the most relevant cloud computing simulators available. In Section 3, the high-level architectural decisions of SCORE are highlighted. The scheduling models under evaluation are described in Section 3.1. In Section 3.3, the core modules of SCORE, which are related to energy efficiency, are described. A number of the parameters available for the configuration of the experimentation are shown in Section 4. In Section 4.2, the workload employed to perform the experimentation in SCORE is characterised. In Section 5, the data available as a result of the experiments performed in SCORE, as well as the means to retrieve this data are explained. The various experimentation scenarios and the result parameters are shown in Section 6. Finally, the conclusions and future work are discussed in Section 7.

2. Related Work

In recent years, many simulators have been developed for the modelling of the main components of the computational cloud systems. In this section, a simple survey and comparative analysis is provided. The following basic trade-offs should be considered when developing a simulation tool: a) Performance versus features; b) Performance versus accessibility; c) Accessibility versus features; and d) Performance versus accuracy.

Although visual general-purpose simulators, such as Insight Maker [6], could be used to simulate computational cloud systems, the development of the model of an energy-efficient cloud-computing infrastructure, such as that described earlier, would be failure-prone and over-sized.

On the other hand, there are other non-general-purpose simulators, such as ElasticTree [7], and CloudSched [8], which are focused on the energy consumption of networking elements and scheduling policies, respectively. Due to this specialisation, these simulators present major limitations and restrictions.

In addition, we evaluated wide-ranging cloud-computing simulation tools, which cover more elements of the Cloud-Computing systems (CC). Each of the frameworks studied simulates different aspects of the CC systems to a different degree of detail. These modelling and implementation decisions render each system different in terms of performance and features provided, which, in turn, makes some of them more suitable for the aforementioned purpose. This class of simulators includes:

Table 1
Simulator comparison.

Simulator	Scheduling models	Energy aware	Energy strategies	Scheduling policies	Performance
GridSim	N	N	N	Y	Medium
CloudSim	N	Y	Y	Y	Medium
GreenCloud	N	Y	Y	Y	Low
Google Omega paper	Y	N	N	N	High
Grid'5000 Toolbox	N	Y	Y	N	High

- **GridSim**. This toolkit [9], based on the SimJava library [10], models various aspects of grid systems, such as users, machines, applications, and networks. However, it fails to consider several cloud-computing features. It also lacks the energy-consumption perspective.
- **CloudSim**. This popular cloud simulator is based on SimJava and GridSim and is mainly focused on IaaS-related operation environment features [11]. It presents a high level of detail, and therefore allows several VM allocation and migration policies to be defined, networking to be considered, features and energy consumption to be taken into account. However, it features certain disadvantages when applied for the simulation of large data-center environments: this high level of detail means that CloudSim is considered cumbersome to execute, especially for data centers composed of thousands to tens of thousands of machines. In addition, it is not principally designed to simulate multiple scheduling models, but takes a largely a monolithic approach.
- **GreenCloud**. This simulator is an extension of the NS2 network simulator. Its purpose is to measure and compute the energy consumption at every data center level, and it pays special attention to network components [12]. However, its packet-level nature compromises performance in order to raise the level of detail, which may be not optimal for the simulation of large data centers. In addition, it is not designed to offer ease of development and extension in various scheduling models.
- **Google Omega lightweight simulator**. This simulator is designed for the comparison of various scheduling models in large clusters. To this end, it focuses on maximizing the performance of the simulations by reducing the level of detail. However, it is not designed to easily develop and extend other scheduling strategies. In addition, this tool fails to consider energy consumption.
- **Grid'5000 Toolbox**. Grid'5000 was built upon a network of dedicated clusters. The infrastructure of Grid'5000 is geographically distributed over various sites, of which the initial 9 are located in France. Grid'5000 Toolbox [13] simulates the behaviour of Grid'5000 resources for real workloads while changing the state of the resources according to several energy policies. The simulator includes:
  a) A GUI that allows the user to perform a set of simulations for each location and to execute a set of energy policies; b) A graphical visualisation of the resources during the simulation, including their states, and future and past jobs; c) A graphical view of the results through several charts and spreadsheets.
  On the other hand, the simulator fails to include various scheduling frameworks and it does not simulate the behaviour or consumption of network devices and resources.

Several of these simulators are well-known, and in-depth comparisons have been presented in the literature [14–17]. In this work, the most important aspects regarding the development and application of power-off/on strategies in order to minimise the energy consumption are presented. Among these:

- **Scheduling models**: This parameter reflects whether the simulation tool has different scheduling models implemented, such as parallel, distributed, and monolithic approaches. In addition, this parameter also considers whether the tool is designed to easily extend and develop new scheduling frameworks, and not only allocation policies.
- **Energy aware**: This parameter reflects whether the simulation tool is capable of measuring and computing energy consumption and efficiency parameters.
- **Shut-down and Power-on policies**: This parameter reflects whether the simulation tool has different shut-down and power-on algorithms implemented. In addition, it also considers whether the tool is designed to easily extend and develop new energy policies.
- **Scheduling strategies**: This parameter reflects whether the simulation tool has different allocation/scheduling algorithms implemented. In addition, it also considers whether the tool is designed to easily extend and develop new strategies.
- **Performance**: This parameter reflects the amount of time for a simulation to be run in a comparable environment. The amount of computational and memory resources is also considered.

A short comparative between the simulation tools described are presented in Table 1.

The simulator described in this work, SCORE, is a high-performance cloud-computing simulation tool focused on energy-efficiency in large data centers. This tool is designed to be easily extended, and offers several strategies already implemented and ready to use, for various scheduling models, allocation, and shut-down and power-on strategies. The high performance

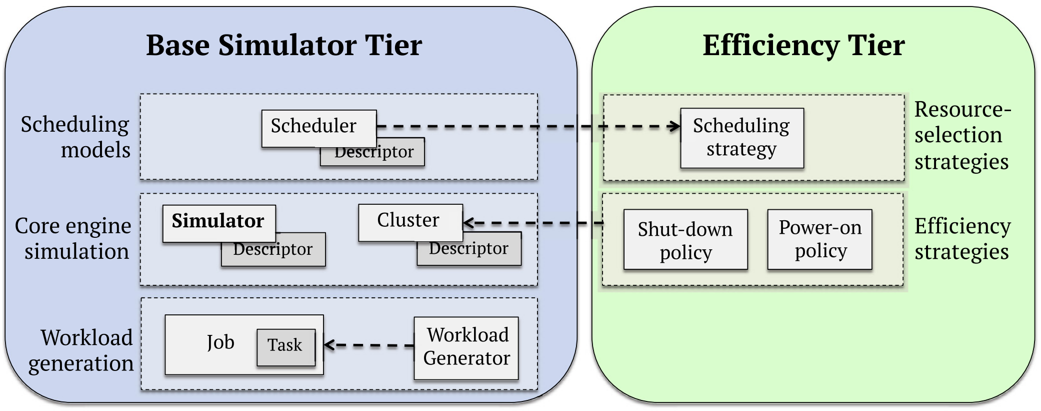


Fig. 1. SCORE architecture.

and ease of use have been achieved by minimising the low-level features, such as networking, low-level machine details, and low-level task details.

3. SCORE Architecture

The SCORE simulator developed herein is based on the model proposed by Robinson in [18]. The main workflow of that simulator can be defined as follows: 1. Definition and initialisation of the problem; 2. Determination of the modeling and general objectives; 3. Identification of the model inputs; 4. Identification of the model outputs; and 5. Determination of the model content and level of detail.

Based on this workflow model in CC, we developed the SCORE architecture as presented in Fig. 1. The SCORE source code is publicly available at:

https://github.com/DamianUS/cluster-scheduler-simulator.

The architectural model is composed of two main modules, namely *the core simulator – (CS)* and *the efficiency module – (EF)*. The core simulator is the core execution engine, and it has been inherited from the Omega lightweight simulator as explained in Section 2. Although the main layers in the *CS* model are based on the same structure present in the original simulator, we made several modifications in order to be able to perform the experimentation related to the security and energy-efficiency. Each experiment is a set of executions and each execution defines all the operational environment details, such as energy policies and scheduling models.

The architecture of *CS* is a 3–layer architecture with the *Workload generation* as the first layer, which is responsible for the generation of the CC workload that will be used in every run of a single experiment. The workload is created only once. This is critical, since it enables the parameters and other aspects used within each run to be reliably compared. The workload is generated by the various data-center utilisation patterns. For instance, the day/night pattern is the predominant pattern in the data centers that execute large web services and applications, since this is common human behaviour.

A workload is composed of a set of *Jobs*. In the same way, a *Job* is composed of a *Bag of Tasks*. A *Task* is the minimum execution unit that may be deployed on a computational server. Each *Task* is mapped to a linux container which is deployed in a similar (but more lightweight) way as a virtual machine (VM). This modern and more flexible virtualisation strategy replaces the traditional virtualisation strategy based on independent virtual machines. Each *Task* deployed on a linux container requires a given amount of computational and memory resources for a given time to be successfully completed. In the current version of this simulator, no linux-container migration nor server consolidation strategy is considered. Thus, once a *Task* is deployed on a server, it runs on this machine until its completion.

The *Core Engine Simulation* layer performs all the simulation duties, reads the workload generated, and performs the scheduling decisions to deploy the tasks on the worker nodes. The *Cluster* is defined by its *Descriptor* and represents the number of computational servers and their features. The current version of SCORE does not provide any networking capabilities due to two main reasons: a) The main goal of this work is to provide a performant and low resource-consuming tool capable of simulating large-scale data centers. These requirements impose serious restrictions regarding the level of detail of the developed features. b) There are several simulation tools that focus on networking details, as stated in Section 2.

Finally, the *Scheduling models* layer implements various scheduling frameworks, such as Omega [2], Mesos [1], and Monolithic models. These schedulers perform the resource-allocation process, and dictate the scheduling decisions to the *Core Engine Simulation* layer.

We developed the *EF* module in order to apply several energy-efficiency and resource-selection strategies. Therefore, this module is logically divided into two layers of the same name. The *Efficiency strategies* layer defines the policies for shutting-down and powering-on computational servers with the objective of optimising the energy consumption of the data

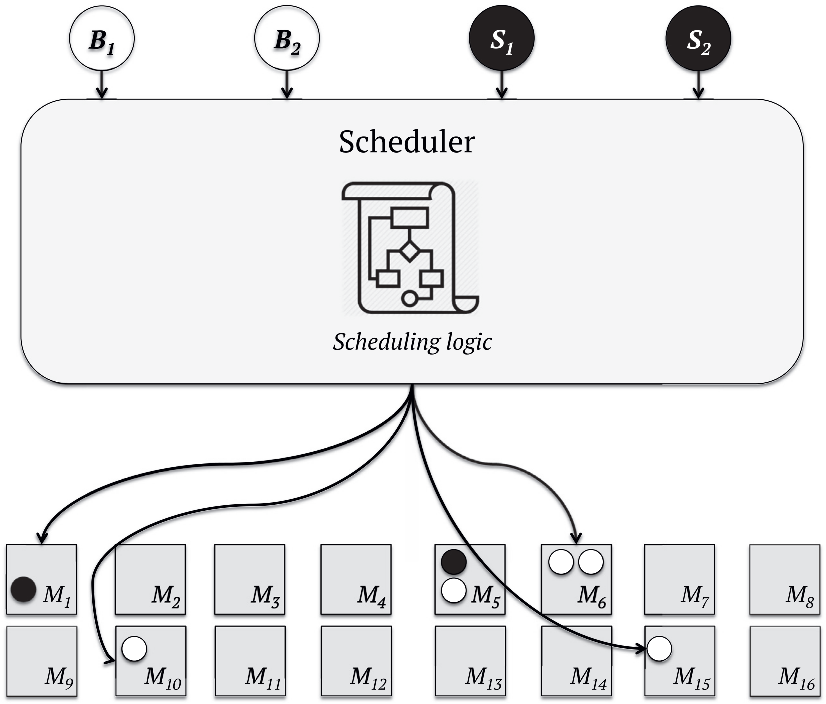


Fig. 2. Monolithic scheduler architecture, B - Batch-type task, S - Service-type task, M - Worker Node.

center. These policies can act over the *Cluster*, changing the energy state of the working nodes from powered-on to shut-down and viceversa. Finally, in the *Resource-selection strategies* layer, several approaches for the reservation of resources are implemented, such as maximizing the dispersion of tasks, deploying them randomly, and minimising the dispersion of tasks. These selection decisions therefore have an impact on the overall performance and energy-saving results.

3.1. Scheduling models

- **Monolithic schedulers**: In this model, a single, centralized scheduling algorithm is employed for all jobs. Google Borg [19] is an example of this kind of scheduling model. The workflow of the monolithic scheduler is demonstrated in Fig. 2.
- **Two-level schedulers**: In this model, the resource allocation and task placement concerns are separated. There is a unique, centralised active resource manager that offers computing resources to multiple parallel, independent, application-level scheduling nodes, as shown in Fig. 3. This approach allows the task-placement logic to be developed for every single application, but also allows the cluster state meta-data to be shared between these schedulers. Mesos [1] is an example of this kind of scheduler.
- **Shared-state schedulers**: In this model, the cluster meta-data is shared between all scheduling agents. The scheduling process is performed by using an out-of-date copy of this shared cluster meta-data. When one of these parallel schedulers performs a scheduling decision based on the probably stale cluster meta-data, the scheduling agent commits the scheduling decision as a transaction in an optimistic way, as shown in Fig. 4. Hence, if any of the scheduling operations committed cannot be applied because the chosen computing resources are no longer free, then that scheduling operation is repeated by the scheduler until no conflicts are found. Google Omega is an example of this kind of model.
- **Fully-distributed schedulers**: In this model, the scheduling frameworks have various independent scheduling nodes which work with a local and out-of-date vision of the cluster state with no central coordination. Sparrow [20] is an example of this kind of scheduler.
- **Hybrid schedulers**: In this model, several scheduling strategies are used (typically a fully distributed architecture is combined with a monolithic or shared-state design) depending on the workload. There are usually two scheduling paths: a distributed path for short or batch tasks, and a centralized path for the remaining tasks. Mercury [21] provides an example of this kind of scheduler.

3.2. SCORE energy-awareness model

SCORE has been developed to enrich the Google Omega Lightweight Simulator. A main feature included is the capability of performing energy-efficiency analysis by applying an energy-consumption model. The CPU is considered in order to compute the energy consumption. The proposed energy-awareness model considers the following states for each CPU core in a

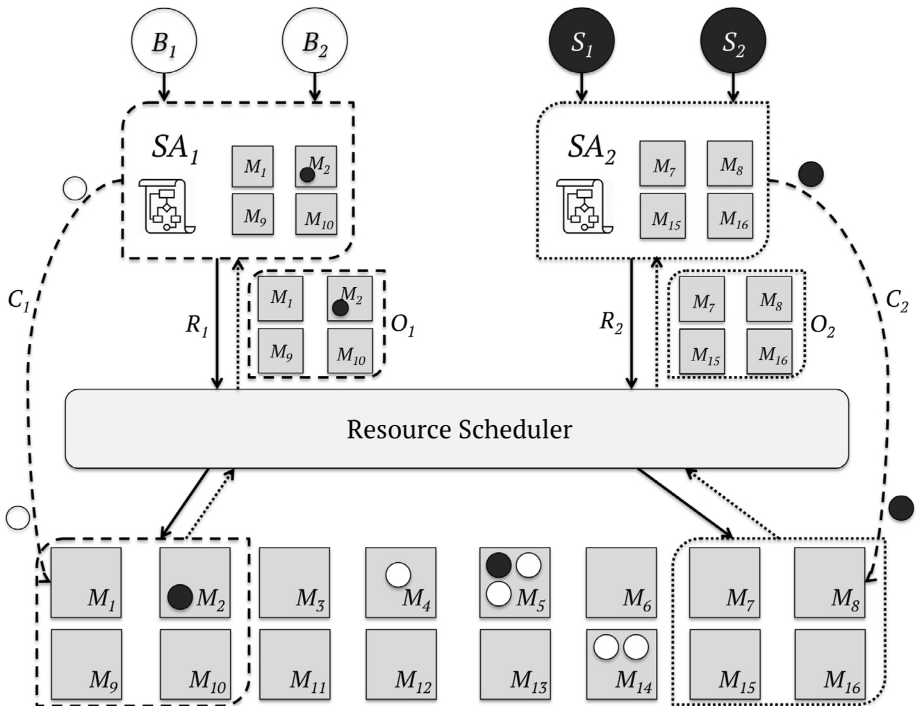


Fig. 3. Two-level scheduler architecture, SA - Scheduler Agent, O - Resource offer, C - Commit.

server: a) *On:* 150W b) *Idle:* 70W. The energy consumption is linearly computed in terms of the utilisation of each CPU core. The following machine power states have also been considered: a) *Off:* 10W b) *Shutting down:* 160W * number of cores c) *Powering on:* 160W * number of cores.

The total energy consumed by the whole data center is measured from time to time by checking the power state of every machine. This time interval is a configurable parameter.

Regarding the shut-down process time parameters, the following values have been assumed: a) $T_{On \to Hibernated}$: 10s, and b) $T_{Hibernated \to On}$: 30s. The power states and transitions are shown in Fig. 5. All the aforementioned power and time parameters can be modified for each experiment.

3.3. SCORE energy-efficiency modules

As aforementioned, the energy-efficiency tier is composed of the following three modules:

- **Shut-down module**: This module is responsible for shutting down the computational servers in order to minimise energy consumption. Several strategies may be used in order to shut down the machines. Each shut-down strategy is implemented in the form of a *Shut-down policy*.
- **Power-on module**: This module is responsible for waking up the machines required to meet present or future workload demands. Several strategies may be used in order to minimise the negative performance impact caused by machines that are not available to immediately execute tasks because they are shut down. Each of these strategies is implemented in the form of a *Power-on policy*.
- **Scheduling module**: This module is responsible for determining which tasks should be deployed on which machine.

3.3.1. Shut-down policies

Power-off policies are responsible for deciding whether or not a machine should be shut down and responsible for triggering the order for the shut-down operation.

In this work, authors have divided the process into making a decision of whether a shut-down action must be taken, and carrying on the actual action of ordering the shut-down of the machine in order to allow various combinations of strategies to be performed. The workflow of this process is illustrated in Fig. 6.

Shut-down decision policies can be: deterministic, such as always shutting-down; or probabilistic, such as shutting down machines following the exponential policy. These decision policies always return a Boolean value which determines whether a given machine must be shut down. In order to make the decision, this policy may check various *Cluster* variables after having finished a task and having freed the resources.

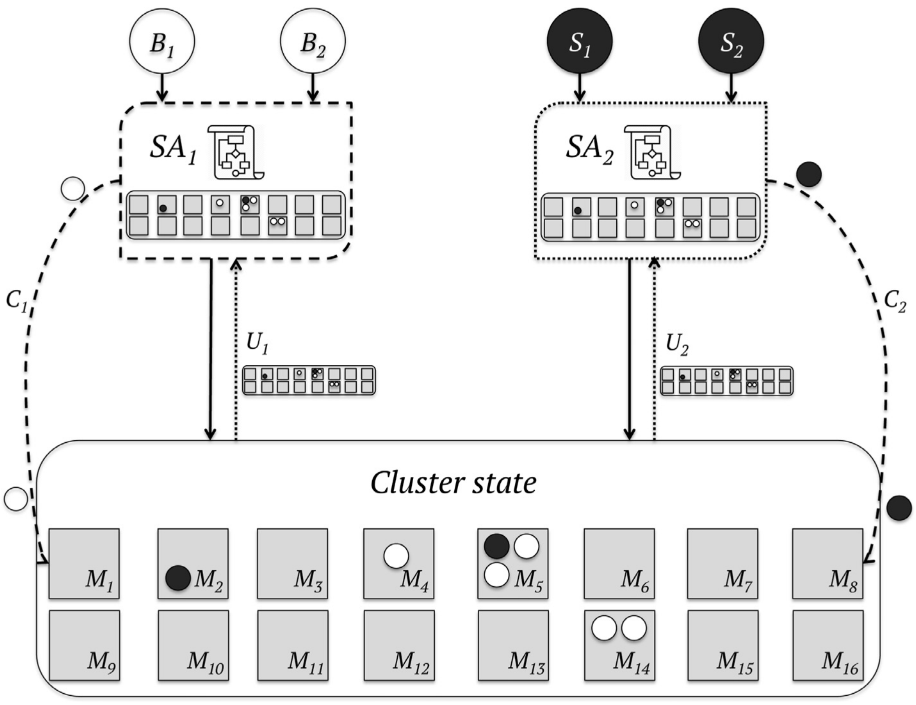


Fig. 4. Shared-state scheduler architecture, U-Cluster State Update.

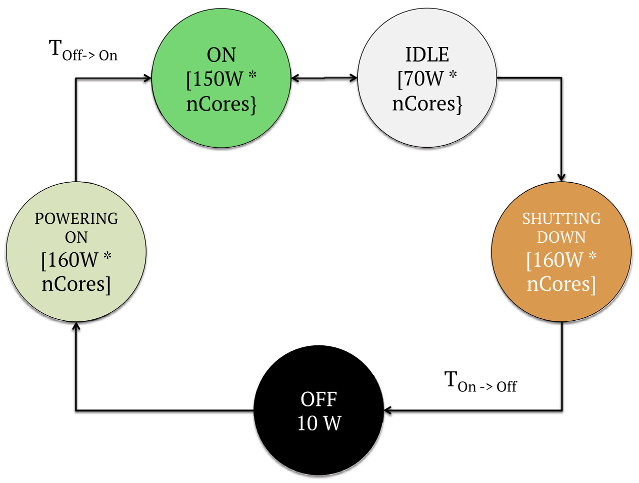


Fig. 5. Machine power states.

Several shut-down policies have been implemented and tested in-depth, as shown in Section 6. These policies include: *Never power off, Always power off, Shut down depending on data-center load, Leave a security margin, Exponential*, and *Gamma*. In addition, various power-off decision policies can be combined by using the logical operators *and* and *or* to achieve policies of a more flexible and complex nature.

3.3.2. Power-on policies

Power-on policies are responsible for maintaining sufficient resources available in order to properly execute the arriving jobs. As the complement to shut-down policies, the strategies developed are critical to guarantee that heterogeneous workloads and peak loads can be executed without causing a negative impact in the overall data-center performance, without breaking SLAs/SLOs, and without affecting the user experience.

As opposed to power-off policies, which make a decision and perform an action independently for each machine, power-on policies work with the overall cell state in order to turn on multiple machines if required by the workload. This power-on

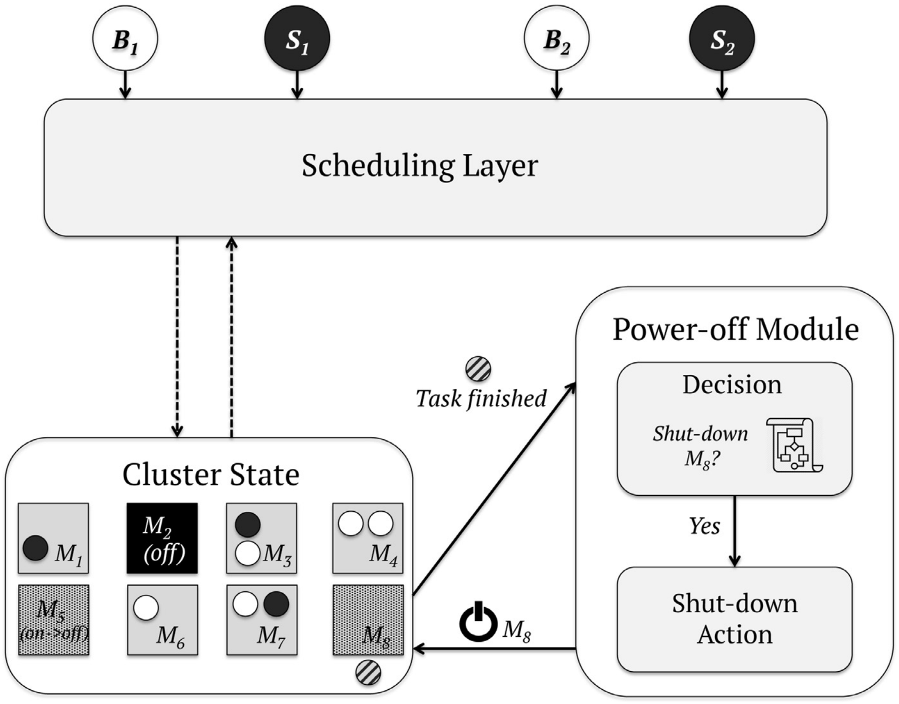


Fig. 6. Power-off module architecture.

business logic is conceptually executed when a new job is scheduled, as opposed to the shut-down business logic, which is executed when resources are released. Of course, each scheduler has different scheduling processes, so powering-on cycles can therefore vary depending on the scheduler. In order to make power-on decisions, the system may require several items of operational information, such as: a) The *Job* that triggered the power-on action; and b) The scheduler model that triggered the power-on action.

In order to be realistic, this simulation tool usually works with a heterogeneous workload with no evident usage pattern. This kind of workload is really complex to predict, and therefore shut-down policies may negatively impact the data-center performance if they are unable to predict near-future workload requirements properly.

Various deterministic and probabilistic power-on decision and action policies have been developed to face this challenge. These policies include: *Never power on, Power-on only the required machines, Power-on a fixed number of machines to maintain a security margin, Power-on a percentage of machines to maintain a security margin*, and power-on policies which make decisions based on statistical distributions, such as *Exponential*, and *Gamma*.

In addition, it is especially interesting that various power-on decision policies may be combined using the logical operators *and* and *or* to achieve policies of a more flexible and complex nature. With this strategy, the benefits of predictive policies may be enjoyed without giving up the possibility of turning on the required machines if the prediction fails or a peak load arrives.

3.3.3. Scheduling strategies

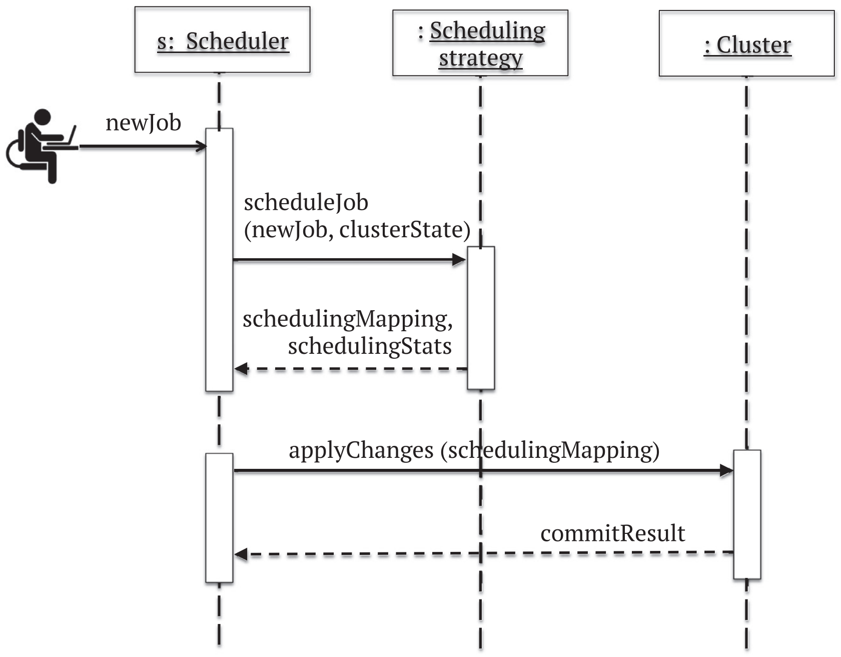
SCORE designs the scheduling strategy as a plug-in piece that is established at the experiment creation time. This scheduling strategy is used by all the schedulers of all scheduling models in order to determine on which machines the tasks should be deployed. Thus, the scheduling strategy works as a black box which uses the information of the whole cluster and of the job for these schedulers, and returns them the mapping between tasks and the machine to be applied. Once this mapping is available, each scheduling model deploys these tasks on the chosen machines depending on their own scheduling logic, as illustrated in Fig. 7.

Several scheduling strategies have been developed. These strategies include: those based on the ETC-matrix genetic process [22], such as *ETC minimising makespan* [23], and *ETC minimising energy* [24], *Random, Spread tasks the maximum, Greedy minimising energy, Greedy minimising makespan, Spread tasks the minimum, Spread tasks the minimum with randomness*.

4. Experiment analysis

4.1. General parameters

Table 2 presents the key parameters of the simulator used in experiments.



s: Scheduler
: Scheduling strategy
: Cluster
newJob
scheduleJob (newJob, clusterState)
schedulingMapping, schedulingStats
applyChanges (schedulingMapping)
commitResult

Fig. 7. Scheduling-strategy workflow.

4.2. Workload parameters

In order to perform realistic experimentation, the workload present in Google traces [25] is considered. Based on the studies made by the research community in [26,27], it is known that realistic jobs are composed of one or more tasks, sometimes thousands of tasks. In addition, two types of jobs are to be considered:

- **Batch jobs**: This workload is composed of jobs which perform a computation and then finish. These jobs have a determined start and end. MapReduce jobs are an example of a *Batch* job.
- **Service jobs**: This workload is composed of long-running jobs which provide end-user operations and infrastructure services. As opposed to *Batch jobs*, these jobs have no determined end. Web servers or services such as BigTable [28] are good examples of a *Service* job.

In order to properly define the workload and create the model for the generated jobs, the following job attributes are considered: a) Inter-arrival time, which represents the time elapsed between two consecutive *Service* jobs or between two *Batch* jobs; b) Number of tasks, which is usually higher for *Batch* jobs than for *Service* jobs; c) Job duration, which may be modified by the machine performance profile; and d) Resource usage, which represents the amount of CPU and RAM that every task in the job consumes [2].

4.2.1. Workload generation

Regarding the generation of the workload, several approaches are implemented and may be used, ranging from uniform generators, followed by several degrees of exponential generators and generators that rely on traces to model the tasks. This workload is generated at the beginning and used for all the experiments of that execution. The following workload generators are available in SCORE:

- **Uniform**: This strategy generates jobs at a uniform rate, of a uniform size and which consume the same amount of resources.
- **Exponential**: This approach generates workloads with jobs that have the inter-arrival time, the number of tasks, and the task duration sampled from exponential distributions. Two versions of the *Exponential* generator have been implemented in order to create both a flat and a day/night patterned workloads.
- **Exponential built from a trace file**: This generator creates workloads where the duration and the number of tasks of all jobs are sampled from exponential distributions built from a trace file. In addition, the *Exponential* and the *Exponential built from a trace file* can be chosen on a per-parameter basis.
- **Trace file**: This approach generates workloads that reproduce a trace file.

Trace-related workload generators require a trace file that contains one job per line. Each line must present the following columns separated by a whitespace: 1. Submission time; 2. Number of tasks; 3. Job duration; 4. Number of CPU cores required by the tasks; and 5. Amount of RAM required by the tasks.

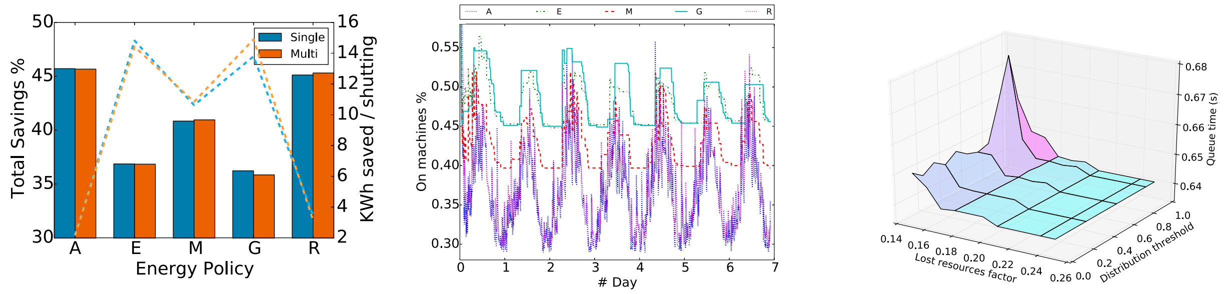
Table 2
Configurable experiment parameters.

Parameter	Description	Values
Cluster	Parameters related to the data center that must be fixed for all experiments	
#Machines	Data-center size	[1 - ∞]
#Cores	Number of CPU cores for every machine	[1 - ∞]
RAM	Amount of RAM in GB for every machine	[0.1 - ∞]
Heterogeneity	Flag to decide whether data-center machines are heterogeneous	Boolean
Machine performance profile	Describe the performance of every machine in the data center. The lower this value, the more performant the server is	Array, size: number of machines [0.01 - ∞]
Machine security profile	Describe the security of every machine in the data center. The higher this value, the more secure the server is [23]	Array, size: number of machines [1 - 5]
Machines energy profile	Describe the energy consumption of every machine in the data center. The lower this value, the more energy-efficient the server is	Array, size: number of machines [0.01 - ∞]
Power-on time	The time required to boot a server, in seconds	[0.1 - ∞]
Shut-down time	The time required to hibernate a server, in seconds	[0.1 - ∞]
Performance	Parameters related to performance iterated in order to create all experiment variations	
Per-job algorithm time	Time spent (in seconds) by the scheduler in order to make a job-level scheduling decision. This simulates the performance of the scheduling algorithm	Array [0.001 - ∞]
Per-task algorithm time	Time spent (in seconds) by the scheduler in order to make a task-level scheduling decision. This simulates the performance of the scheduling algorithm	Array [0.001 - ∞]
Blacklist	The percentage of machines not to be used	Array [0.0 - ∞]
Inter-arrival	This parameter rewrites the inter-arrival time generated for all jobs, replacing it with a fixed time instead	Array [0.001 - ∞]
Energy	Parameters related to performance iterated in order to create all experiment variations	
Shut-down policies	The shut-down policies to be run, such as: *Always power off, Exponential*, and *Gamma*	Array
Power-on	The power-on policies to be run	Array
Scheduling	The scheduling strategies to be run	Array
Sorting	These strategies are used by greedy scheduling strategies to sort the candidate servers for their later selection	Array
Specific	Parameters used by specific schedulers	
Schedulers assigned to workloads	Mapping that describes how many and which schedulers are assigned to which workload type (Batch/Service) used by non-monolithic schedulers. Each row adds a new scheduler to serve a workload	Map [Scheduler name -> Workload name]
Conflict mode	Approach used by non-monolithic schedulers to decide whether a *commit* results in conflict	resource-fit sequence-numbers
Transaction mode	Approach used by non-monolithic schedulers to decide what to do when a *commit* results in conflict	all-or-nothing incremental

5. Information retrieval

The results of the experiments are stored in one to several Google protocol buffer files. The main blocks of these protocol buffer files include the following:

- **Experiment Environment** section stores the information related to the global configuration of the experiments.
- **Experiment Results** section stores the information related to global results for each of the experiments performed.
- **Workload Stats** section stores the workload-specific information for both *Batch* and *Service* jobs for each *Experiment Result*, including the details of the genetic process when used.
- **Scheduler Stats** section stores the scheduler-specific information for each *Experiment Result*, including daily and workload-related details.



(a) Energy savings vs. kWh saved per shut-down

(b) # Powered-on machine evolution for multiple policies

(c) Queue-time detail for Exponential policy

Fig. 8. Simulation results graphic scripts.

Table 3
Experiment outputs.

Parameter	Description	Values
Performance	Output parameters related to the data center overall and per-workload performance	
Queue time until first deploy	Represents the time a job waits in the queue until its first task is scheduled (in seconds).	[0.0 - ∞]
Queue time until full deploy	Represents the time a job waits in the queue until it is totally scheduled (not completion).	[0 - ∞]
Timed-out jobs	Number of jobs left unscheduled after 100 unsuccessful scheduling tries for a given job or 1000 tries for any given task in a job.	[0.0 - ∞]
Scheduler occupation	Percentage of scheduler utilisation on average	[0.0 - 100.0]
Job scheduling attempts	Number of scheduling operations needed to fully deploy a job.	[0 - ∞]
Task scheduling attempts	Number of tasks scheduling operations needed to fully deploy a job.	[0 - ∞]
Energy-efficiency	Output parameters related to the data-center resource and energy efficiency.	
Energy consumed	Total data-center energy consumption (in kWh)	[0.0 - ∞]
Energy saved vs. current system	Total energy saved by applying energy-efficiency policies compared to the same scenario with no energy-efficiency policies applied (in kWh).	[0.0 - ∞]
Shut-downs	Number of shut-down operations.	[0 - ∞]
Idle resources	Percentage of resources operating in an *Idle* state on average.	[0.0 - 100.0]
KWh saved per shut-down operation	This represents the energy saved against the number of shut-downs performed. It shows the *goodness* of the shut-down operations performed.	[0.0 - ∞]

- **Efficiency Stats** section stores the energy-efficiency parameters for each *Experiment Result*.
- **Measurements** section stores the performance and energy-efficiency metrics gathered in each measurement performed every few seconds for each *Experiment Result* in order to show the cluster evolution.

The information stored in the protocol buffer files can't be visualized directly. Hence, a set of python scripts, aimed to create valuable human-readable and graphic information from these files, have been implemented. The results of some of these graphic scripts are illustrated in Fig. 8.

5.1. Output indicators

Table 3 presents the most relevant results indicators of the experimentation performed in terms of performance and energy-efficiency.

6. Examples of usage

In this section, a set of experiments have been run in order to illustrate certain experiment parameterisation and results related to both performance and energy-efficiency. Although each experiment may show a different subset of parameters and results, all the parameters are used and returned in each experiment. In addition, several parameters have been fixed in order to keep the tests simple, such as always using the default power-on policy, which aims to power on machines when a workload cannot be served.

Each experiment runs for a 7-day operation period and processes a day-night patterned workload that uses, on average, approximately 30% of the resources, with load peaks achieving approximately 60% of data-center utilisation. The parameters of the jobs in this workload are generated by using an exponential distribution for the inter-arrival time, number of tasks, and duration, while the security constraints are generated randomly. Regarding these constraints, the following values are used: a) *Batch* jobs are composed of 50 tasks and *Service* of 9 tasks on average; b) *Batch*-job tasks take 90 seconds and *Service*-job tasks 2,000 seconds to finish on average; and c) *Batch*-job tasks consume 0.3 CPU cores and 0.5 GB of memory, while *Service*-job tasks consume 0.5 CPU cores and 1.2 GB of memory.

The data center is composed of 1,000 machines of 4 CPU cores and 8GB RAM. The energy, performance, and security profiles of these machines are randomly generated.

6.1. Genetic process experimentation

In this test, a set of experiments focused on illustrating certain results of the genetic process of the ETC-matrix-based scheduling strategies are run by using: a) a monolithic scheduling model, and b) fixed algorithm times.

The results corresponding to *Batch* jobs and the *Single-path monolithic scheduler* are presented in Table 4, where it can be observed that the genetic scheduling strategy focused on minimising the makespan achieves better performance results, and it is shown how this makespan average evolves between the genetic-process epochs.

6.2. Performance experimentation for the Omega scheduler

Several parameters are available for the evaluation of the impact on the operation environment of the energy-efficiency policies and algorithm performance. A number of these parameters are presented after having executed a new set of experiments by using the following simulation configuration: a) the Omega scheduling model, with 4 schedulers responsible for serving *Batch* jobs and 1 scheduler for *Service* jobs; b) one scheduling strategy which strives to maximise machine usage while minimising resource contention; and c) the *Resource-fit* conflict mode.

Table 4
Parameters of the minimising-makespan genetic process.

Shut-down	Scheduling	Savings (%)	Makespan Avg. (s)	Epoch 0 (s)	Epoch 100 (s)
Never off	Makespan	N/A	236.01	324.02	202.07
Never off	Energy	N/A	290.55	N/A	N/A
Random	Makespan	55.88	258.41	344.51	273.99
Random	Energy	55.31	311.91	N/A	N/A

Table 5
Omega performance experimentation.

Shut-down policy	Transaction mode	Per-job alg. time (s)	Per-task alg. time (ms)	Savings (%)	Queue time until first deploy (ms)	Queue time until full deploy (ms)
Never off	all-or-nothing	0.1	10	N/A	0.0	0.0
Never off	all-or-nothing	0.1	100	N/A	560.0	1,021.4
Never off	all-or-nothing	1.0	10	N/A	0.2	0.2
Never off	all-or-nothing	1.0	100	N/A	791.8	1,604.0
Never off	incremental	0.1	10	N/A	0.0	0.0
Never off	incremental	0.1	100	N/A	12.9	27.5
Never off	incremental	1.0	10	N/A	0.0	0.0
Never off	incremental	1.0	100	N/A	18.5	38.9
Always off	all-or-nothing	0.1	10	46.08	0.1	0.8
Always off	all-or-nothing	0.1	100	40.86	2,294.7	5,368.4
Always off	all-or-nothing	1.0	10	44.97	1.2	2.2
Always off	all-or-nothing	1.0	100	38.78	3,503.1	9,061.6
Always off	incremental	0.1	10	45.06	0.1	0.5
Always off	incremental	0.1	100	45.10	36.9	84.6
Always off	incremental	1.0	10	45.02	1.1	2.5
Always off	incremental	1.0	100	45.05	46.8	107.9

Table 6
Mesos energy-efficiency experimentation.

Shut-down policy	Per-job alg. time (s)	Per-task alg. time (ms)	Energy consumed (kWh)	Energy Saved (kWh)	# shut downs	Idle resources (%)
Never off	0.1	10	57,259	0	0	69.30
Never off	0.1	100	57,359	0	0	69.29
Never off	1.0	10	57,372	0	0	69.27
Never off	1.0	100	57,440	0	0	69.28
Always off	0.1	10	31,138	25,774	18,520	5.97
Always off	0.1	100	31,748	25,208	18,473	7.28
Always off	1.0	10	31,642	25,321	19,719	7.00
Always off	1.0	100	31,808	25,137	17,550	7.48
Random	0.1	10	32,003	24,949	9,136	7.97
Random	0.1	100	32,694	24,295	8,968	9.55
Random	1.0	10	32,702	24,225	8,816	9.82
Random	1.0	100	32,739	24,241	9,606	9.65
Exponential	0.1	10	34,952	21,842	1,156	16.43
Exponential	0.1	100	36,119	20,823	1,286	18.33
Exponential	1.0	10	35,953	20,951	1,214	17.87
Exponential	1.0	100	36,513	20,408	1,816	19.49

The results corresponding to *Batch* jobs are presented in Table 5, where it can be observed that the algorithm performance and the conflict-handling strategy have a notable impact both on queue times and on energy consumption.

6.3. *Energy-efficiency experimentation for the Mesos scheduler*

In addition to performance parameters, the energy-efficiency parameters constitute the core of the simulation tool. In order to illustrate these parameters, several experiments have been run by using the same configuration as laid out in Section 6.2 for the *Omega* scheduler.

The results corresponding to *Batch* jobs are presented in Table 6, where it can be observed that the shut-down policies exert a major impact on energy consumption and hardware stress, and that a high level of energy-efficiency can be achieved without performing many shut-down operations, thereby minimising the performance impact.

7. Summary and future work

Our simulation tool: SCORE is presented as an extension to the Google Omega lightweight simulator, a simulator focused on the comparison of the performance between scheduling models, especially parallel frameworks in large-scale data centers. The model of the Google Omega lightweight simulator has been enhanced with extensions and improvements for: a) The development of an energy consumption model; b) The extension and creation of allocation policies; c) The extension and creation of energy-efficiency policies based on shutting down and powering on machines; d) The addition of heterogeneity to data center machines; and e) The consideration of security profiles.

It has been shown that the application of all the features of this tool to heterogeneous workloads in large-data centers results in major potential improvements from both the energy-efficiency and performance point of view.

As future work, several aspects of the tool are being improved. These improvements include:

- Greater ease of use and extendability of the implemented code.
- Addition of a visual interface to set the experiment configurations and to execute the experiments.
- Addition of a real-time visualizer of the power and performance state of the machines.
- Incorporation of support for other workload constraints, such as time and precedence constraints.
- Tasks dependency, which implies, among others, networking considerations.
- Development of *Tasks* migration and server consolidation techniques.

Acknowledgement

The research is supported by the VPPI - University of Sevilla. This article is based upon work from COST Action IC1406 “High-Performance Modelling and Simulation for Big Data Applications” (cHiPSet), supported by COST (European Cooperation in Science and Technology).

References

[1] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R.H. Katz, S. Shenker, I. Stoica, Mesos: A platform for fine-grained resource sharing in the data center., NSDI 11 (2011) 22.

[2] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, J. Wilkes, Omega: flexible, scalable schedulers for large compute clusters, in: Proceedings of the 8th ACM European Conference on Computer Systems, ACM, 2013, pp. 351–364.

[3] J. Kołodziej, Evolutionary hierarchical multi-criteria metaheuristics for scheduling in large-scale grid systems, 419, Springer, 2012.
[4] J. Koomey, Growth in data center electricity use 2005 to 2010, A report by Analytical Press, completed at the request of The New York Times 9 (2011).
[5] A. Fernández-Montes, D. Fernández-Cerero, L. González-Abril, J.A. Álvarez-García, J.A. Ortega, Energy wasting at internet data centers due to fear, Pattern Recognit. Lett. 67 (2015) 59–65.
[6] S. Fortmann-Roe, Insight maker: A general-purpose tool for web-based modeling & simulation, Simulation Model. Practice Theory 47 (2014) 28–45.
[7] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown, Elastictree: Saving energy in data center networks., in: Nsdi, 10, 2010, pp. 249–264.
[8] W. Tian, Y. Zhao, M. Xu, Y. Zhong, X. Sun, A toolkit for modeling and simulation of real-time virtual machine allocation in a cloud data center, IEEE Trans. Autom. Sci. Eng. 12 (1) (2015) 153–161.
[9] R. Buyya, M. Murshed, Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency Comput. 14 (13-15) (2002) 1175–1220.
[10] F. Howell, R. McNab, Simjava: A discrete event simulation library for java, Simul. Series 30 (1998) 51–56.
[11] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software 41 (1) (2011) 23–50.
[12] D. Kliazovich, P. Bouvry, S.U. Khan, Greencloud: a packet-level simulator of energy-aware cloud computing data centers, J. Supercomput. 62 (3) (2012) 1263–1283.
[13] A. Fernández-Montes, L. Gonzalez-Abril, J.A. Ortega, L. Lefèvre, Smart scheduling for saving energy in grid computing, Expert Syst. Appl. 39 (10) (2012) 9443–9450.
[14] K. Bahwaireth, E. Benkhelifa, Y. Jararweh, M.A. Tawalbeh, et al., Experimental comparison of simulation tools for efficient cloud and mobile cloud computing applications, EURASIP J. Inf. Secur. 2016 (1) (2016) 1–14, doi:10.1186/s13635-016-0039-y.
[15] W. Tian, M. Xu, A. Chen, G. Li, X. Wang, Y. Chen, Open-source simulators for cloud computing: Comparative study and challenging issues, Simul. Model. Pract. Theory 58 (2015) 239–254.
[16] C. Badii, P. Bellini, I. Bruno, D. Cenni, R. Mariucci, P. Nesi, Icaro cloud simulator exploiting knowledge base, Simul. Model. Pract. Theory 62 (2016) 1–13.
[17] G. Kecskemeti, Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds, Simul. Model. Pract. Theory 58 (2015) 188–218.
[18] S. Robinson, Conceptual modelling for simulation part ii: a framework for conceptual modelling, J. Oper. Res. Soc. 59 (3) (2008) 291–304.
[19] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at google with borg, in: Proceedings of the Tenth European Conference on Computer Systems, ACM, 2015, p. 18.
[20] K. Ousterhout, P. Wendell, M. Zaharia, I. Stoica, Sparrow: distributed, low latency scheduling, in: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, ACM, 2013, pp. 69–84.
[21] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G.M. Fumarola, S. Heddaya, R. Ramakrishnan, S. Sakalanaga, Mercury: Hybrid centralized and distributed scheduling in large shared clusters., in: USENIX Annual Technical Conference, 2015, pp. 485–497.
[22] A. Jakobik, D. Grzonka, J. Kolodziej, H. González-Vélez, Towards secure non-deterministic meta-scheduling for clouds, in: 30th European Conference on Modelling and Simulation, ECMS 2016, Regensburg, Germany, May 31 - June 3, 2016, Proceedings., 2016, pp. 596–602, doi:10.7148/2016-0596.
[23] A. Jakobik, D. Grzonka, F. Palmieri, Non-deterministic security driven meta scheduler for distributed cloud organizations, Simul. Model. Practice Theory (available online 4 November 2016). https://doi.org/10.1016/j.simpat.2016.10.011.
[24] A. Jakóbik, D. Grzonka, J. Kołodziej, Security supportive energy aware scheduling and scaling for cloud environments, in: European Conference on Modelling and Simulation, ECMS 2017, Budapest, Hungary, May 23-26, 2017, Proceedings., 2017, pp. 583–590, doi:10.7148/2017-0583.
[25] C. Reiss, J. Wilkes, J.L. Hellerstein, Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release, in: 3rd International Workshop on Cloud Management (CLOUDMAN), IEEE, Maui, HI, USA, 2012, pp. 1279–1286.
[26] O.A. Abdul-Rahman, K. Aida, Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon, in: IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Singapore, 2014, pp. 272–277.
[27] S. Di, D. Kondo, C. Franck, Characterizing cloud applications on a Google data center, in: 42nd International Conference on Parallel Processing (ICPP), Lyon, France, 2013.
[28] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber, Bigtable: A distributed storage system for structured data, ACM Trans. Comput. Syst. (TOCS) 26 (2) (2008) 4.