Putting the Next 500 VM Placement Algorithms to the Acid Test

Adrien Lebre, Jonathan Pastor, Mario Südholt
Ascola Research Group
Inria, Lina, Mines Nantes
Nantes, France
firstname.lastname@inria.fr

ABSTRACT

Most current infrastructures for cloud computing leverage static and greedy policies for the placement of virtual machines. Such policies impede the optimal allocation of resources and the satisfaction of operational guarantees like service-level agreements. In recent years, more dynamic and often more efficient policies based, e.g., on consolidation and load balancing techniques, have been developed. Due to the underlying complexity of cloud infrastructures, these policies are evaluated either using limited scale testbeds/in-vivo experiments or ad-hoc simulator techniques. These validation methodologies are unsatisfactory for two important reasons: they (i) do not model precisely enough real production platforms (size, workload representativeness, failure, etc.) and (ii) do not enable the fair comparison of different approaches. More generally, new placement algorithms are thus continuously proposed without really identifying the significant benefits of each of them.

In this article, we show how VMPlaceS, a dedicated simulation framework enables researchers (i) to study and compare VM placement algorithms , (ii) to detect possible limitations at large scale and (iii) easily investigate different design choices. Built on top of the SimGrid simulation platform, VMPlaceS provides programming support to ease the implementation of placement algorithms and runtime support dedicated to load injection and execution trace analysis. We investigate three well known strategies in terms of reactivity and fault tolerant properties. Diving into the details, we also identify several modifications that can significantly increase their performance. We believe that VMPlaceS will allow researchers to validate the significant benefits of new placement algorithms, thus accelerating VM placement research results and favouring the transfer to IaaS production platforms.

CCS Concepts

•General and reference → Evaluation; Validation; •Information systems → Data centers; •Networks → Cloud computing; •Theory of computation → *Scheduling algorithms;* **•Computing methodologies → Simulation tools; Simulation evaluation; •Software and its engineering → Software performance;**

Keywords

Cloud Computing, VM Placement strategies, Simulation

1. INTRODUCTION

Even if more flexible and often more efficient approaches to the Virtual Machine Placement Problem (VMPP) have been developed, most of the popular Cloud Computing (CC) system management [1, 2, 6], *a.k.a.* IaaS toolkits [25], continue to rely on elementary Virtual Machine (VM) placement policies that prevent them from maximizing the usage of CC resources while guaranteeing VM resource requirements as defined by Service Level Agreements (SLAs). Typically, a batch scheduling approach is used: VMs are allocated according to user requests for resource reservations and tied to the nodes where they were deployed until their destruction. Besides the fact that users often overestimate their resource requirements, in particular for web-services and entreprise information systems [10, 29], such static policies are definitely not optimal for CC providers, since the effective resource requirements of each operated VM may significantly vary during its lifetime.

An important impediment to the adoption of more advanced strategies such as consolidation, load balancing and other SLA-ensuring algorithms that have been deeply investigated by the research community [16, 18, 20, 27, 32, 33] is related to the experimental processes that have been used to validate them: most VMPP proposals have been evaluated either by leveraging ad-hoc simulators or small testbeds. These evaluation environments are not accurate and not representative enough to (i) ensure their correctness on real platforms and (ii) perform fair comparisons between them. Implementing each proposal and evaluating it on representative testbeds in terms of scalability, reliability and varying workload changes would definitely be the most rigorous way to observe and propose appropriate solutions for CC production infrastructures. However, *in-vivo* (*i.e.*, real-world) experiments, if they can be executed at all, are always expensive and tedious to perform (for recent reference see [8]). They may even be counterproductive if the observed behaviors are clearly different from the expected ones. Consequently, new placement algorithms are continuously proposed without really identifying the significant benefits of each of them.

Middleware'15 Submitted to the Experimentation And Deployment track
ACM ISBN TBD.
DOI: TBD

In this article, we illustrate in details how VMPlaceS, a dedicated simulation framework that enables in-depth investigations and fair comparisons of VM placement algorithms [23], can help researchers to mesure and validate the advantages of new proposals.

Built on top the SimGrid toolkit [14], VMPlaceS allows users to study large-scale scenarios that include server crashes and that involve tens of thousands of VMs, each executing a specific workload that evolves during the simulation lifetime. Such conditions are mandatory to perform simulations that are representative enough of CC production platforms [9, 10, 29]. We chose to base VMPlaceS on SimGrid since (i) the latter's relevance in terms of performance and validity has already been demonstrated [4] and (ii) because it has been recently extended to integrate virtual machine abstractions and an accurate live migration model [19].

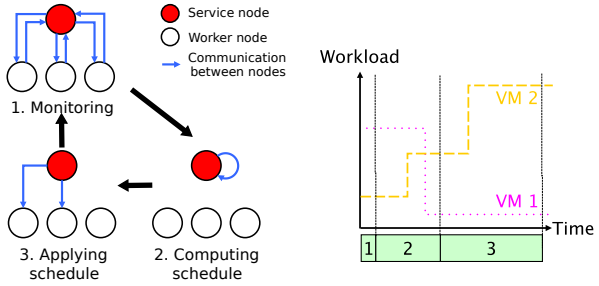
In addition to validating the accuracy of VMPlaceS, we discussed in our previous work [23] a preliminary analysis of three well-known VMPP approaches: Entropy [18], Snooze [16], and DVMS [27]. Besides being well-known from the literature, we chose these three systems as they are built on three different software architecture approaches: Entropy relies on a centralized model, Snooze on a hierarchical one and DVMS on a fully distributed one. In this article, we complete this preliminary analysis by diving into details, showing the relevance of VMPlaces to compare VM placement strategies, to identify major limitations and investigate variants that can have a significant impact on the overall behavior. in particular, we show in this article:

- The importance of the duration of the reconfiguration phase (*i.e.*, the step where VMs are relocated throughout the infrastructure) in comparison to the computation one (*i.e.*, the step where the scheduler try to solve the VMPP, see Section 2 for a complete definition).
- The high number of migrations overall.
- The pros and cons of partitioning an infrastructure into small or large groups to handle the scheduling process;
- The relevance of a reactive strategy in comparison to a periodic one.
- The moderate impact of node crashes for the hierarchical approach even in the presence of high failure rates.
- The relevance of VMPlaceS to identify the best VM placement approaches that have the potential to handle CC production infrastructures.

The rest of the article is structured as follow. Section 2 highlights the importance of the scalability, reliability and reactivity properties for the VM Placement problem. Section 3 gives an overview of the SimGrid framework on which our proposal is built. Section 4 introduces VMPlaceS and discusses its general functioning. The three algorithms implemented as use-cases are presented in Section 5 and evaluated in Section 6. Section 7 and Section 8 present, respectively, related work as well as a conclusion and future work.

2. THE VM PLACEMENT PROBLEM

The VMPP can be summarized in three-steps (see Figure 1(a)): monitoring the resources usages, computing a new schedule each time is it needed and applying the resulting reconfiguration plan (*i.e.*, performing VM migration and suspend/resume operations to switch to the new placement solution).



(a) Scheduling steps

(b) Workload fluctuations during scheduling

Figure 1: VM scheduling Phases

VMPP solutions stand and fall with their scalability, reliability and reactivity of properties (*i.e.*, the time to solve a SLA violations), because they have to maintain a placement that satisfies the requirements of all VMs while optimizing the usage of CC resources. For instance, a naive implementation of a master/worker approach as described in Figure 1(a) would prevent workload fluctuations to be taken into account during the computation and the application of a schedule, potentially leading to artificial violations (*i.e.*, resource violations that are caused by the VMPP mechanism). In other words, the longer each phase, the higher the risk that the schedule may be outdated when it is computed or eventually applied (see the different loads during the three phases in Figure 1(b)). Similarly, servers and network crashes can impede the detection and resolution of resource violations if the master node crashes or if a group of VMs is temporarily isolated from the master node.

VMPP solutions can only be reasonably evaluated if their behavior in the presence of such adverse events can be analyzed. Providing a framework that facilitates such studies and increases their reproducibility is the main objective of VMPlaceS.

3. SIMGRID, A GENERIC TOOLKIT

We now briefly introduce the toolkit on which VMPlaceS is based. SimGrid is a toolkit for the simulation of potentially complex algorithms executed on large-scale distributed systems. Developed for more than a decade, it has been used in a large number of studies described in more than 100 publications. Its main characteristics are the following:

- Extensibility: after Grids, HPC and P2P systems, SimGrid has been recently extended with abstractions for virtualization technologies (*i.e.*, Virtual Machines including a live migration model [19]) to allow users to investigate Cloud Computing challenges [24].
- Scalability: it is possible to simulate large-scale scenarios; as an example, users can simulate applications composed of 2 million processors and an infrastructure composed of 10,000 servers [14].

- Flexibility: it enables simulations to be run on arbitrary network topologies under dynamically changing computations and available network resources.

- Versatile APIs: users can leverage SimGrid through easy-to-use APIs for C and Java.

To perform simulations, users should develop a *program* and define a *platform* file and a *deployment* file. The *program* leverages, in most cases, the SimGrid MSG API that allows end-users to create and execute SimGrid abstractions such as processes, tasks, VMs and network communications. The *platform* file provides the physical description of each resource composing the environment and on which aforementioned computations and network interactions will be performed in the SimGrid world. The *deployment* file is used to launch the different SimGrid processes defined in the *program* on the different nodes. Finally, the execution of the program is orchestrated by the SimGrid engine that internally relies on an constraint solver to correctly assign the amount of CPU/network resources to each SimGrid abstraction during the entire simulation.

SimGrid provides many other features such as model checking, the simulation of DAGs (Direct Acyclic Graphs) or MPI-based applications. In the following, we only give a brief description of the virtualization abstractions that have been recently implemented and on which VMPlaceS relies on (for further information regarding SimGrid see [14]).

The VM support has been designed so that all operations that can be performed on a host can also be performed inside a VM. From the point of view of a SimGrid Host, a SimGrid VM is an ordinary task while from the point of view of a task running inside a SimGrid VM, a VM is considered as an ordinary host. SimGrid users can thus easily switch between a virtualized and non-virtualized infrastructure. Moreover, thanks to MSG API extensions, users can control VMs in the same manner as in the real world (*e.g.*, create/destroy VMs; start/shutdown, suspend/resume and migrate them). For migration operations, a VM live migration model implementing the precopy migration algorithm of Qemu/KVM has been integrated into SimGrid. This model is the only one that successfully simulates the live migration behavior by taking into account the competition arising in the presence of resource sharing as well as the memory refreshing rate of the VM, thus determining correctly the live migration time as well as the resulting network traffic [19]. These two capabilities were mandatory to build the VM placement simulator toolkit.

4. VMPLACES

The purpose of VMPlaceS is to deliver a generic tool to evaluate new VM placement algorithms and offer the possibility to compare them. Concretely, it supports the management of VM creations, workload fluctuations as well as node apparitions/removals. Researchers can thus focus on the implementation of new placement algorithms and evaluate how they behave in the presence of changes that occur during the simulation. VMPlaceS has been implemented in Java by leveraging the messaging API (MSG) of SimGrid. Although the Java layer has an impact of the efficiency of SimGrid, we believe its use is acceptable because Java offers important benefits to researchers for the implementation of advanced scheduling strategies, notably concerning the ease of implementation of new strategies. As examples, we reimplemented the Snooze proposal in Java and the DVMS proposal using Scala/Java.

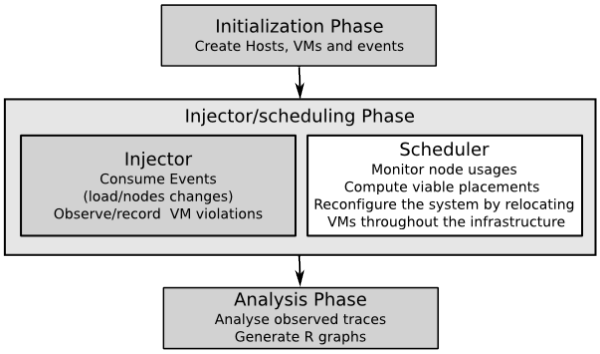


Figure 2: VMPlaceS's Workflow
Gray parts correspond to the generic code while the white one must be provided by end-users. The current version is released with three different schedulers (centralized/hierarchical and distributed).

In the following we give an overview of the framework and describe its general functioning.

4.1 Overview

From a high-level view, VMPlaceS performs a simulation in three phases: (i) initialization (ii) injection and (iii) trace analysis (see Figure 2). The initialization phase corresponds to the creation of the environment, the VMs and the generation of the queue of events that may represent, *e.g.*, load changes. The simulation is performed by at least two SimGrid processes, one executing the *injector* and a second one executing the to-be-simulated *scheduling algorithm*. During the simulation the scheduling strategy is evaluated by taking into account the different events played by the injector. Currently, the supported events are VM CPU load changes and node apparitions/removals that we use to simulate node crashes. The last phase consists in the analysis of the collected traces in order to gather the results of the simulation, notably by means of the generation of figures representing, *e.g.*, resource usage statistics.

Regarding the non-generic part, users must develop their scheduling algorithm by leveraging the SimGrid messaging API and a more abstract interface that is provided by VMPlaceS and consists of the classes `XHost`, `XVM` and `SimulatorManager` classes. The two former classes respectively extend SimGrid's `Host` and `VM` abstractions while the latter controls the interactions between the different components of the VM placement simulator. Throughout these three classes, users can inspect, at any time, the current state of the infrastructure (*i.e.*, the load of a host/VM, the number of VMs hosted on the whole infrastructure or on a particular host, check whether a host is overloaded, etc.)

4.2 Initialization Phase

In the beginning, VMPlaceS creates n VMs and assigns them in a round-robin manner to the first p hosts defined in the platform file. The default platform file corresponds to a cluster of $h + s$ hosts, where h corresponds to the number of hosting nodes and s to the number of services nodes.

The values n, h and s constitute input parameters of the simulations (specified in a Java property file). These hosts are organized in form of topologies, a cluster topology being the most common ones. It is possible, however, to define more complex platforms to simulate, for instance, scenarios involving federated data centers.

Each VM is created based on one of the predefined VM classes. A VM class corresponds to a template specifying the VM attributes and its memory footprint. Concretely, it is defined in terms of five parameters: the number of cores `nb_cpus`, the size of the memory `ramsize`, the network bandwidth `net_bw`, the maximum bandwidth available to migrate it `mig_speed` (generally equal to `net_bw`) and the maximum memory update speed `mem_speed` when the VM is consuming 100% of its CPU resources. As pointed out in Section 3, the memory update speed is a critical parameter that governs the migration time as well as the amount of transferred data. By giving the possibility to define VM classes, VMPlaceS allows researchers to simulate different kinds of workload (*i.e.*, memory-intensive vs non-intensive workloads), and thus analyze more realistic Cloud Computing problems. Available classes are defined in a specific text file that can be modified according to the user's needs. Finally, all VMs start with a CPU consumption of 0 that will evolve during the simulation in terms of the injected load as explained below.

Once the creation and the assignment of VMs completed, VMPlaceS spawns at least two SG processes, the *injector* and the launcher of the selected scheduler. The first action of the *injector* consists in creating the different event queues and merge them into a global one that will be consumed during the second phase of the simulation. For now, we generate two kinds of event: *CPU load* and *node crash* events.

Regarding *CPU loads*, the event queue is generated in order to change the load of each VM every t seconds on average. t is a random variable that follows an exponential distribution with rate parameter λ_t while the CPU load of a VM evolves according to a Gaussian distribution defined by a given mean (μ) as well as a given standard deviation (σ). t, μ and σ are provided as input parameters of a simulation. Although a recent study on the characterization of production CC workloads [10] advocates the use of exponential distributions to capture both CPU usages and the temporal variability, we believe that a Gaussian distribution is more appropriate to simulate small periods of CC platforms (*i.e.*, less than 2 hours). In addition to being more expressive for researchers (*i.e.*, it is simpler to define μ and σ than finding the right λ), the exponential law does not allow VMPlaceS to concentrate the possible changes around a common value for a particular period, which corresponds to the normal behavior of CC workloads at a certain moment of the day [29]. In other words, the load of a CC workload significantly differs at the scale of the day but not at the scale of the hour. Finding the right distribution that will enable VMPlaceS to simulate CC production platforms over longer periods (*i.e.*, days) is part of our planned future work. Getting back to the Gaussian, as the CPU load can fluctuate between 0 and 100%, VMPlaceS prevents the assignment of nonsensical values when the Gaussian distribution returns a number smaller than 0 or greater than 100. Although this has no impact on the execution of the simulation, we emphasize that this can reduce/increase the effective mean of the VM load, especially when σ is high. Hence, it is important for users to specify appropriate values.

Regarding *node crashes*, the event queue is generated in order to turn off a node every f seconds on average for a duration of d seconds [9]. Similarly to the t value above, f follows an exponential distribution with rate λ_f. f and d are also provided as input parameters of a simulation.

We underline that adding new events can easily be done by simply defining new event Java classes implementing the `InjectorEvent` interface and by adding the code in charge of generating the associated queue. Such a new queue will be merged into the global one and its events will then be consumed similarly to other ones during the *injector phase*. We are currently integrating new events in order to simulate network and disk usage events.

Finally, we highlight that each random process used in VMPlaceS is initialized with a seed defined in a configuration file. This way, we can ensure that different simulations are reproducible and may be used to establish fair comparisons.

4.3 Injector Phase

Once the VMs and the global event queue are ready, the evaluation of the scheduling mechanism can start. First, the injector process iteratively consumes the different events that represent, for now, load changes of a VM or turning a node off or on. Changing the load of a VM corresponds to the creation and the assignment of a new SimGrid task in the VM. This new task has a direct impact on the time that will be needed to migrate the VM as it increases or decreases the current CPU load and thus its memory update speed. When a node is turning off, the VMs that were running on that node are temporarily discarded, *i.e.*, they are hidden and cannot be accessed until the node comes back to life. This way, the scheduler cannot handle them. We leave for future work other approaches that can better match realistic scenarios such as turning off the VMs and reprovisioning them on other nodes.

As defined by the scheduling algorithm, VMs will be suspended/resumed or relocated on the available hosts to meet scheduling objectives and SLA guarantees. Note that users must implement the algorithm in charge of solving the VMPP but also the code in charge of applying reconfiguration plans by invoking the appropriate methods available from the `SimulatorManager` class. This step is essential as the reconfiguration cost is a key element of dynamic placement systems.

Last but not least, it is noteworthy that VMPlaceS really invokes the execution of each scheduling strategy in order to get the effective reconfiguration plan. That is, the computation time that is observed is not simulated but corresponds to the effective one, only the workload inside the VMs and the migration operations are simulated in SimGrid. It is hence mandatory to propagate the reconfiguration time into the SimGrid engine.

4.4 Trace Analysis

The last step of VMPlaceS consists in analyzing the information that has been collected during the simulation. This analysis is done in two steps. First, VMPlaceS records several metrics related to the platform utilization throughout the simulation by leveraging an extended version of SimGrid's TRACE module[1]. This way, visualization tools that

[1] http://simgrid.gforge.inria.fr/simgrid/3.12/doc/tracing.html

have been developed by the SimGrid community, such as PajeNG [3], may be used. Furthermore, our extension enables the creation of a trace file in the JSON file format, which is used to generate several figures using the R statistical environment [11] about the resource usage during the simulation.

By default, VMPlaceS records the load of the different VMs and hosts, the appearance and the duration of each violation of VM requirements in the system, the number of migrations, the number of times the scheduler mechanism has been invoked and the number of times it succeeds or fails to resolve non-viable configurations. Although these pieces of information are key elements to understand and compare the behavior of the different algorithms, we emphasize that the TRACE API enables the creation of as many variables as necessary, thus allowing researchers to instrument their own algorithm with specific variables that record other pieces of information.

5. DYNAMIC VMPP ALGORITHMS

To illustrate the interest of VMPlaceS, we implemented three dynamic VM placement mechanisms respectively based on the Entropy [18], Snooze [16], and DVMS [27] proposals. For the three implementations, we chose to use the latest VMPP solver that has been developed as part of the Entropy framework [17].

Giving up consolidation optimality in favor of scalability, this algorithm provides a "repair mode" that enables the correction of VM requirement violations. The algorithm considers that a host is overloaded when the VMs try to consume more than 100% of the CPU capacity of the host. In such a case, the algorithm looks for an optimal viable configuration until it reaches a predefined timeout. The optimal solution is a new placement that satisfies the requirements of all VMs while minimizing the cost of the reconfiguration. Once the timeout has been triggered, the algorithm returns the best solution among the ones it finds and applies the associated reconfiguration plan by invoking live migrations in the simulation world.

Although using the Entropy VMPP solver implies a modification from the original Snooze proposal, we highlight that our goal is to illustrate the capabilities of VMPlaceS and thus we believe that such a modification is acceptable as it does not change the global behavior of Snooze. Moreover by conducting such a comparison, we also investigate the pros and cons of the three architecture models on which these proposals rely on (*i.e.*, centralized, hierarchical and distributed).

Before discussing the simulation results, we describe in this section an overview of the three implemented systems. We highlight that the extended abstractions for hosts (`XHost`) and VMs (`XVM`) as well as the available functions of the SimGrid MSG API enabled us to develop them in a direct and natural manner.

5.1 Entropy-based Centralized Approach

The centralized VM placement mechanism consists in one single SimGrid process deployed on a service node. This process implements a simple loop that iteratively checks the viability of the current configuration by invoking every p seconds the aforementioned VMPP solver. p is defined as an input parameter of the simulation.

As the Entropy proposal does not provide a specific mechanism for the collect of resource usage information but simply uses an external tool (namely ganglia), we had two different ways to implement the monitoring to process: either by implementing additional asynchronous transmissions as a real implementation of the necessary state updates would proceed or, in a much more lightweight manner, through direct accesses by the aforementioned process to the states of the hosts and their respective VMs. While the latter does not mimic a real implementation closely, it can be harnessed to yield a valid simulation: overheads induced by communication in the "real" implementation, for instance, can be easily added as part of the lightweight simulation. We have implemented this lightweight variant for the monitoring

Regarding fault tolerance, similarly to the Entropy proposal, our implementation does not provide any failover mechanism.

Finally, as mentioned in Section 4.4, we monitor, for each iteration, whether the VMPP solver succeeds or fails. In case of success, VMPlaceS records the number of migration that has been performed, the time it took to apply the reconfiguration and whether the application of the reconfiguration plan led to new violations.

5.2 Snooze-based Hierarchical Approach

We now present Snooze [16] as a second case study of how to implement and simulate advanced algorithms. We present its architecture summarizing its main characteristics from its original presentation [16] and additional information stemming from personal communications of the Snooze developers and its implementation [5, 31].

5.2.1 Architecture

Snooze harnesses a hierarchical architecture in order to support load balancing and fault tolerance, cf. Fig. 3.

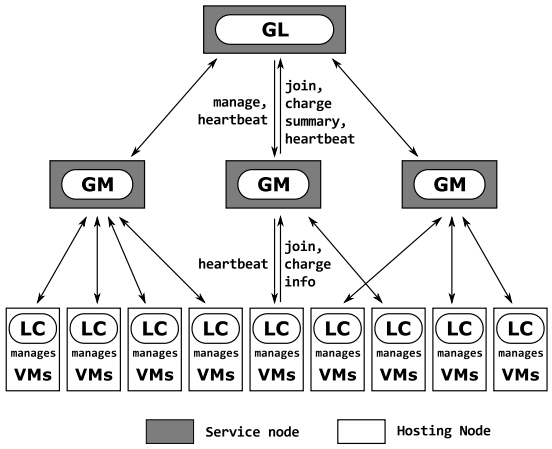


Figure 3: Overview of Snooze's architecture

At the top of the hierarchy, a *group leader (GL)* centralizes information about the whole cluster using summary data about *group managers (GMs)* that constitute the intermediate layer of the hierarchy. GMs manage a number of *local controllers (LCs)* that, in turn, manage the VMs assigned to nodes. The GL and the GMs are deployed on service nodes while the LCs are executed on hosting node. During execution, higher-level components periodically send heartbeats to lower-level ones; monitoring information, *e.g.*, about the

system load, is also sent periodically in the opposite direction. In order to propagate information down the hierarchy, Snooze relies on hardware support for multicast communication. Finally, a number of replicated entry points allows clients to contact the GL, *e.g.*, in order to submit new VMs for integration into the system.

Simulation using VMPlaceS. The `XHOST`, `XVM` and `SimulatorManager` classes have been harnessed to implement the core architectural abstractions (*i.e.*, VM monitoring and manipulations), the remaining concepts and algorithms of Snooze have been implemented using Simgrid's primitives and standard Java mechanisms. Communication between Snooze actors is implemented based on Simgrid's primitives for, mainly asynchronous, event handling. Hardwaresupported multicast communication that is used, *e.g.*, to relay heartbeats, is implemented as a dedicated actor that manages a state representing GL and GM heartbeat groups and relaying heartbeat events. Finally, our Snooze simulation uses, as its original counterpart, a multi-threaded implementation (*i.e.*, based on multiple SG processes) in order to optimize reactivity even for large groups of LCs (or GMs) that have to be managed by one GM (or GL).

5.2.2 Algorithms

Apart from the handling of faults (described below), two types of algorithms are of major importance for the administration of the Snooze architecture: the algorithms that enable components to dynamically enter the system and the algorithms that propagate info between the components.

A GL is created, if it does not exist, by promotion of a GM that is selected according to some leader election algorithm. When a GM joins a cluster, it starts listening on a predefined channel for the heartbeat of the GL and registers once it has received the heartbeat. New LCs first also wait for the GL heartbeat, contact the GL then in order to obtain a GM assignment, and finally register at the GM assigned to them.

Two kinds of (load) information are passed within the system: the periodic heartbeat message sent by the GL and the GMs; second, periodic load information sent from LCs to their respective GMs and summary load info sent by the GMs to the GL.

5.2.3 Fault tolerance

GLs, GMs and LCs may fail during the system execution. System components identify that a node on the corresponding higher-level node has failed (the GL in case of a GM, a GM in the case of an LC) in an asynchronous fashion through the lack of heartbeat messages.

In the case of a GL failure, one of the GMs becomes the new GL, stops its GM activities and prevents the LCs it manages so that they can start rejoining the system. If a GM fails, the GL and the LCs it has managed will become aware of it based on the lack of heartbeats, update its data structures and, for the LCs, rejoin the system. If an LC fails, its GM will finally learn of it due to the missing heartbeat and charge information of the LC. The GM will then remove the LC from its data structures.

5.3 DVMS-based Distributed Approach

As the third use-case, we have implemented the DVMS (Distributed Virtual Machine Scheduler) proposal for the cooperative and fully distributed placement of VMs [27].

5.3.1 Architecture

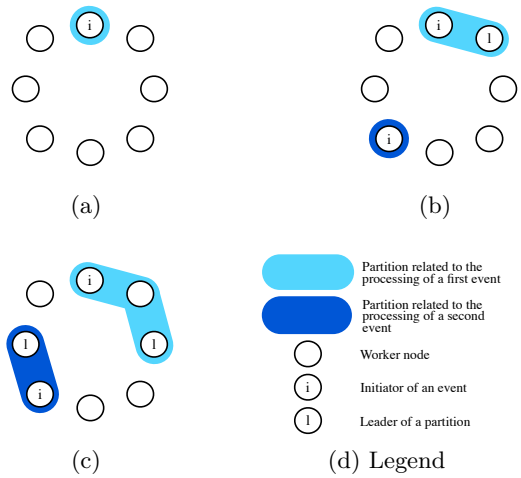


Figure 4: Processing two events simultaneously

A DVMS agent is deployed on each node in order to manage the VMs on the node and collaborate with (the agents of) neighboring nodes. Agents are defined on top of an overlay communication network, which defines the node-neighbor relation and can be structured (using, *e.g.*, Chord [30]) or unstructured. For this study, we have implemented a simple but effective unstructured overlay that enables the agents to collaborate without side effects: when necessary, *e.g.*, in case of node failures, the overlay provides a link to a neighbor of a node on the latter's request.

When a server is overloaded (*i.e.*, VMs hosted on the server require more resources than available), an *Iterative Scheduling Procedure (ISP)* is started: a partition initially containing only the overloaded node is created; the partition then grows by including free nodes until the resource requirements can be satisfied by a VM reconfiguration. This way, resource problems that appear at many different nodes can be handled in parallel using different partitions.

Each partition includes two special nodes, the initiator and the leader. The initiator of a partition is its initial node (i.e. the overloaded node). The leader of a partition is the node that was the last to be added to the partition: it manages the scheduling computations necessary to resolve the overload resource conflict. If a valid reconfiguration plan cannot be computed, a new node will be inserted in the partition, who becomes the new leader of the partition.

5.3.2 Iterative Scheduling Procedure

When a node N_i detects that it cannot provide enough resources for its hosted VMs, it generates a partition and reserves itself to solve the problem (see Figure 4(a)) and thus initiates a partition. Then, its closest neighbor, as defined by the network overlay, is considered.

If this neighbor, N_{i+1}, is already part of another partition, its neighbor is considered. Otherwise, N_{i+1} joins the partition (see Figure 4(b)). If the partition is not valid anymore (*e.g.*, because the workload of the partition's VM has decreased), N_{i+1} cancels the reservations, destroys the partition and thus frees its nodes for another problem solving procedure. On the contrary, if the procedure is still valid, N_{i+1} notifies members of the partition that it has become the new leader. The other nodes then send it information

about their capacities and current load. The leader, in turn, starts a scheduling computation looking for a reconfiguration within the current partition. If no solution is found, the same algorithm is applied to the next node N_{i+2}. In the extreme case a partition may grow until all resources in a cluster contribute to the resolution of its resource scheduling problem. This approach harnesses as few nodes as possible, thus accelerating the scheduling computations to the maximum possible.

5.3.3 Fault-tolerance

The main advantage of using overlay networks is that they have built-in fault tolerance mechanisms. DVMS therefore works on top of an overlay network such as Chord: when a node needs to rebalance its VMs workload, it uses the overlay network to find collaborators. For this study we implemented a simple overlay network as a flat list of agents: a typical request for collaborators includes the list of agents that are already collaborating with the requesting agent. A link to a new collaborator is then provided to the requesting agent. Communication is performed by message exchanges containing immutable data: our implementation harnesses the principles of the actor model in order to ease the handling of concurrency and distributed issues.

Harnessing the fault tolerance mechanisms of the underlying overlay network is, however, not sufficient. If the leader of a partition crashes, a new leader must take over in order for the resource problem to be solved and the nodes of a partition to be finally freed. To avoid these issues, DVMS now relies on timeout mechanisms. Each node of a partition periodically checks whether the state of its partition changed recently (*e.g.*,, if a new node joined the partition) and can thus identify if the partition's leader is not active anymore. In this case, each node leaves the partition and can be integrated in other partitions.

6. EXPERIMENTS

In this section, we discuss the results of the simulations we performed on the Entropy, Snooze and DVMS strategies. First, we present a general study analyzing the violation times, the duration of the computation and reconfiguration phases and the number of migration performed for the standard implementations (*a.k.a.* vanilla code). Second, we examine some variants of and improvements to Snooze and DVMS that have been studied easily thanks to VMPlaceS. We highlight that the accuracy of the results have been validated in our previous work [23].

6.1 Comparing Entropy, Snooze and DVMS (Vanilla Impl.)

6.1.1 Experimental Conditions

All simulations have been performed on the Grid'5000 testbed [7]. Each execution was running on a dedicated server, thus avoiding interferences between simulations and ensuring reproducibility between the different invocations.

VMPlaceS has been configured in order to simulate an homogeneous infrastructure of PMs composed of 8 cores, 32 GB of RAM and 1 Gpbs Ethernet NIC. To enable fair comparisons between the three strategies, the scheduling resolver only considered 7 cores (*i.e.*, one was devoted to run the Snooze GL or the DVMS processes). Dedicating one core for the host OS and other administrative processes is something which is quite common and, as we believe, acceptable as part of our experimental methodology.

We conducted simulations in order to study infrastructures composed of 128, 256, 512, 1024, for each PM number hosting 10 times as many VMs. Additional simulated PMs have been provided to execute the Entropy and Snooze service nodes on distinct nodes. For Snooze, one GM has been created for every 32 LCs (*i.e.*, PMs). Entropy and Snooze are invoked every 30 seconds. Finally, it is noteworthy that no service node had to be provisioned for DVMS as a DVMS process is executed directly on top of the hosting nodes.

In order to cope with real DC conditions, we defined the parameters for node crashes to simulate a fault on average every 6 months for a duration of 300 seconds. These values correspond to the Mean Time To Failure (MTTF) and the Mean Time To Repair (MTTR) of a Google DC server [9, pp. 107-108]. We underline that at the scale we performed our simulations such a crash ratio was not sufficient to impact the behavior of the scheduling policies. Dedicated simulations were mandatory to study the influence of node crashes with an higher failure rate (see Section 6.2.3)

Regarding the virtual machines, ten VMs have been initially launched on each simulated PM. Each VM has been created as one of the 8 VM predefined classes. The template was 1:1GB:1Gbps:1Gbps:X, where the memory update speed X was a value between 0 and 80% of the migration bandwidth (1Gbps) in steps of 10. Starting from 0%, the load of each VM varied according to the exponential and the Gaussian distributions previously described. The parameters were λ = No VMs/300 and μ= 60, σ = 20. Concretely, the load of each VM varied on average every 5 min in steps of 10 (with a significant part between 40% and 80%). The stationary state was reached after 20 min of the simulated time with a global load of 85% as depicted in Fig. 5(a). To accelerate the simulations, we have chosen to limit the simulated time to 1800 seconds. It is noteworthy that the consolidation ratio, *i.e.*, the number of VMs per node, has been defined to generate a sufficient number of violations. We discovered that under a global load of 75%, the simulated infrastructure almost did not face VM violations with our selected Gaussian distribution. Such a result is rather satisfactory as it can explain why most production DCs target such an overall utilization rate.[2]

All configuration files used to perform the discussed simulations can be downloaded from the VMPlaceS repository.[3]

6.1.2 General Analysis

Fig. 5(b) presents the cumulated violation time for each placement policy while Tables 1, 2 and 3 give more details by presenting the mean and the standard deviations of the duration of, respectively, the violations, the computation and reconfiguration phases. As anticipated, the centralized approach did not scale and became almost counterproductive for the largest scenario in comparison to a system that did not use any dynamic scheduling strategy. The more nodes Entropy has to monitor, the less efficient it is on both the computation and reconfiguration phases. Regarding the computation, the VMPP is a NP-Hard problem and thus it is not surprising that it takes more time to resolve larger

[2] http://www.cloudscaling.com/blog/cloud-computing/amazons-ec2-generating-220m-annually/

[3] http://beyondtheclouds.github.io/VMPlaceS/

Infrastructure Size	Algorithm		
	Centralized	Hierarchical	Distributed
128 nodes	23.30 ± 15.64	22.62 ± 15.27	9.47 ± 2.49
256 nodes	39.06 ± 26.89	24.03 ± 15.30	9.49 ± 2.28
512 nodes	60.67 ± 49.66	25.45 ± 25.18	9.56 ± 2.62
1024 nodes	87.62 ± 90.67	29.31 ± 35.55	9.67 ± 2.37

Table 1: Duration of violations ($Med \pm \sigma$)

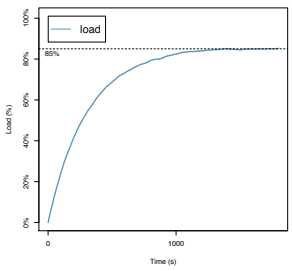
Infrastructure Size	Algorithm		
	Centralized	Hierarchical	Distributed
128 nodes	5.01 ± 8.38	2.70 ± 4.86	0.03 ± 0.02
256 nodes	10.08 ± 16.96	3.32 ± 5.28	0.02 ± 0.02
512 nodes	16.41 ± 29.32	2.86 ± 4.92	0.01 ± 0.01
1024 nodes	27.60 ± 52.99	3.35 ± 5.28	0.01 ± 0.02

Table 2: Duration of computations ($Med \pm \sigma$)

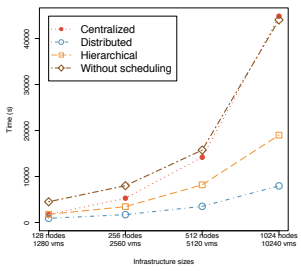
problems. Regarding the reconfiguration, as Entropy has to solve much more violations simultaneously, the reconfiguration plan is more complex for large scenarios, including several migrations coming from and going to the same nodes. Such reconfiguration plans are non optimal as they increase the bottleneck effects at the network level of each involved PM. Such a simulated result is valuable as it confirms that reconfiguration plans should avoid as much as possible such manipulations.

Regarding Snooze, whose performance is better than those of Entropy, we may erroneously conclude that the hierarchical approach is not competitive compared to the distributed strategy at the first sight. However, diving into the details, we can see that both the computation and reconfiguration phases are almost of constant duration (around 3 seconds and 10 seconds) and not much longer than DVMS's corresponding phases, especially for the reconfiguration phase, which is predominant. These results can be easily explained: the centralized policy adresses the VMPP by considering all nodes at each invovation, while the hierarchical and the distributed algorithms divide the VMPP into sub problems, considering smaller numbers of nodes (32 PMs in Snooze and 4 nodes on average with DVMS). These results raises the question of the influence of the group size, *i.e.*, the ratio of LCs attached to one GM, on Snooze's performance (a relationship we investigate in Section 6.2).

Last but not least, Table 4 presents the number of migrations that has been performed overall for each simulation. While DVMS enables the resolution of violations in a short



(a) Infrastructure load



(b) Cumulated Violation Time

Figure 5: Simulation Results - 10 VMs per node (VM load: $\mu = 60$ and $\sigma = 20$)

Infrastructure Size	Algorithm		
	Centralized	Hierarchical	Distributed
128 nodes	10.44 ± 1.95	10.02 ± 0.14	10.01 ± 0.10
256 nodes	11.10 ± 2.95	10.16 ± 1.22	10.00 ± 0.00
512 nodes	12.22 ± 5.26	10.15 ± 1.19	10.01 ± 0.12
1024 nodes	20.64 ± 9.87	10.22 ± 1.33	10.03 ± 0.41

Table 3: Duration of reconfigurations ($Med \pm \sigma$).

Infrastructure Size	Algorithm		
	Centralized	Hierarchical	Distributed
128 nodes	103	70	93
256 nodes	135	164	194
512 nodes	163	267	344
1024 nodes	266	651	842

Table 4: Number of migrations.

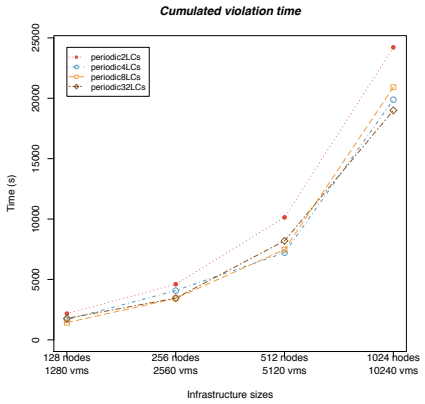
time, one of its potential drawbacks is the number of migration it performs. In addition to adding a significant overhead at the network level, performing migrations can negatively impact the performance of the workload running inside the VM during the relocation phases. Hence, it is usually better to try to find the right trade-off between the reactivity criterion and the number of VM migration as investigated by other VM placement strategies [15]. Diving into the details, VMPlaceS monitors how many times each VM has been migrated. This feature is also interesting as several schedulers, including the ones discussed in the present article, do not consider whether a VM has been previously migrated or not. Hence, it is possible to migrate the same VM a significant number of times while never relocating the others. As we have illustrated here, VMPlaceS enables researchers to monitor such a metric.

6.1.3 *Simulations Scalability*

To conclude this general comparison, although the simulations discussed in this article are limited to 10K VMs, we succeeded to conduct DVMS simulations including up to 8K PMs/80K VMs in a bit less than two days. Similarly, we performed DVMS simulations for 10K VMs over a simulated period of 3600 and 7200 seconds (*i.e.*,. two hours). The duration to get the results were respectively 7420 and 17198 seconds. We do not discuss into details these results for the hierarchical approach a la Snooze because it was not possible to run a sufficient number of simulations at such scale. The Snooze protocol being more complex than the DVMS one (heartbeats, consensus, ...), the time to perform a similar experiment is much more important (around 7 days for the 80K VMs experiment). The time-consuming portions of the code are related to SimGrid internals such as `sleep` and `send/recv` calls. Hence, we have contacted the SimGrid core developers in order to investigate how we can reduce the required time to perform such advanced simulations.

6.2 Exploring Variants and Possible Improvements

In the following, we present several variants of the placement algorithms introduced in Sec. 5 that have been discussed in the literature or come up during the implementation of their models using VMPlaceS. This section provides strong evidence that the modification and evaluation of existing algorithms is much facilitated by our simulation framework.



(a) Total Violation Times

Infra. Size	No. of failed reconfigurations			
	2 LCs	4 LCs	8 LCs	32 LCs
128	19	0	0	0
256	29	0	0	0
512	83	1	0	0
1024	173	7	0	0

(b) No. of Failed Reconfigurations

Infra. Size	Algorithm			
	2 LCs	4 LCs	8 LCs	32 LCs
128	0.16 ± 1.23	0.34 ± 1.81	0.58 ± 2.40	2.53 ± 4.62
256	0.18 ± 1.31	0.42 ± 1.99	0.66 ± 2.50	2.65 ± 4.69
512	0.15 ± 1.20	0.33 ± 1.78	0.67 ± 2.54	2.83 ± 4.98
1024	0.19 ± 1.37	0.42 ± 2.02	0.89 ± 2.90	2.69 ± 4.91

(c) Means ± Std deviations of computation durations.

Figure 6: Hierarchical placement: influence of varying group sizes

6.2.1 *Varying group sizes*

As a first example, we propose to clarify the influence of the group size observed during the general comparison. Concretely, we performed additional simulations aiming at investigating whether a smaller group size for Snooze can lead to similar performances of DVMS and reciprocally, whether DVMS can increase its reactivity by integrating more nodes at the first iteration of each ISP. We highlight that the use of VMPlaceS eased such a study as it has consisted to simply relaunch the previous simulation with a distinct assignment for Snooze and to slightly modify the ISP algorithm for DVMS.

Figure 6 presents the simulated values obtained for the Snooze scenarios with 2, 4, 8 and 32 LCs per GM for four infrastructure sizes. The overall performance (*i.e.*, cumulated violation time), shown in Fig. 6(a), shows that 2 LCs per GM result in significantly higher violation times. All other group sizes yield violation times that are relatively close, which indicates that a small group size does not help much in resolving violations faster.

The relatively bad performance of the smallest group size can be explained in terms of the number of failures of the reconfiguration process, that is, overloading situations that are discovered but cannot be resolved because the GM managing the overloaded VM(s) did not dispose of enough resources, see Table 6(b). Groups of 2 LCs per GM are clearly insufficient at our load level (60% mean, 20% stddev). Failed reconfigurations are, however, already very rare in the case of 4 LCs per GM and do not occur at all for 8 and 32 LCs per GM. This is understandable because the load is statistically evenly distributed among the LCs and tthe load profile we evaluated only rarely results in many LCs of a GM to be overloaded. Violations can therefore be resolved even in the case of a smaller number (4) LCs available for load distribution.

Conversely, we can see that the duration of the overall reconfiguration phases decreases strongly along with the group size. It reaches a value close to the computation times of DVMS for a group size of 4-LCs per GM, see Fig. 6(c). We thus cannot minimize computation times and violation times by reducing the number of LCs because larger group sizes are necessary to resolve overload situations if the VM load gets higher. Once again, this information is valuable as it will help researchers to design new algorithms favoring the automatic discovery of the optimal subset of nodes capable to solve violations under for given load profiles.

In contrast, DVMS resolves this trade-off by construction because of its automatic and dynamic choice of the partition size necessary to handle an overloaded situation. However, it can be interesting to directly add more than one node to form the first group, *a.k.a.* microcosm in the DVMS terminology, especially because having only two nodes seems to be not enough as depicted by the previous Snooze scenarios.

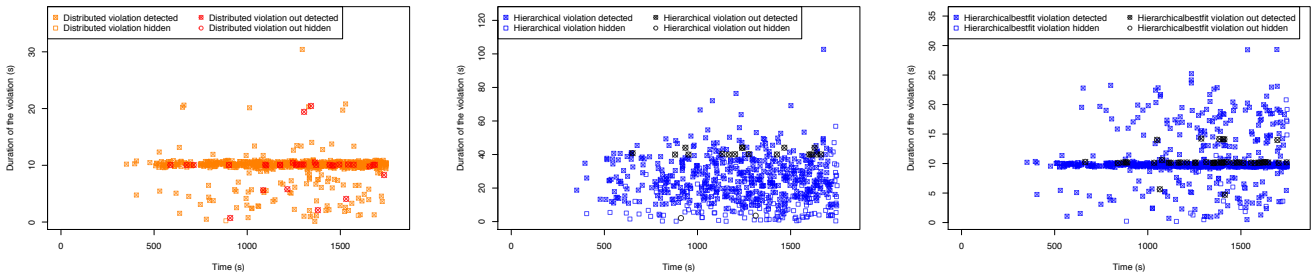
Infrastructure Size / flavour	Violation time (s)		No. migrations	
	Original	4 nodes	Original	4 nodes
128 nodes	9.47 ± 2.49	9.39 ± 3.08	93	106
256 nodes	9.49 ± 2.28	9.61 ± 2.35	194	182
512 nodes	9.56 ± 2.62	9.65 ± 2.62	344	315
1024 nodes	9.67 ± 2.37	9.95 ± 2.41	842	802

Table 5: Comparison of two DVMS flavours.

Table 5 presents the violation time and the number of migrations for the DVMS vanilla implementation and our variant that directly integrates four nodes at the first step of the ISP. In addition to having a similar violation time, it is noticeable that the number of migrations is slightly smaller in the DVMS variant for most cases (understanding why the 128 nodes leads to more migrations and a larger deviation requires more investigations). This gain in terms of migrations is due to the quality of reconfiguration plans: as more nodes can be used to rebalance the VMs workload, its efficiency is improved.

6.2.2 *Hierarchical scheduling: periodic vs. reactive*

Our vanilla implementation of Snooze [16] schedules VMs in a periodic fashion as introduced before. Using VMPlaceS, we have also developed an alternative, reactive, strategy to scheduling: as soon as resource conflicts occur, LCs avert their GMs of them; the GMs then immediately initiate scheduling. Implementing this reactive scheme can be done using our framework in two manners. First, by implementing additional asynchronous communication of the necessary state updates as a real implementation would proceed. Second,



(a) DVMS (b) Snooze Periodic (c) Snooze Reactive

Figure 7: Details of violations duration occurring during simulation (1024 PMs).

in a more lightweight manner through direct accesses by the GMs to the states of their respective LCs. In order to ensure that this lightweight implementation mimics a real implementation closely, delays induced by communication in the "real" implementation are accounted for explicitly (congestion issues are not relevant in this case because notification of a resource conflict implies little communication and conflict resolution blocks the GM and its LCs anyway). We have implemented this lightweight variant of reactive scheduling including an explicit model of communication delays. Using the abstractions provided by VMPlaceS, reactive scheduling has been implemented by adding or modifying just 4 lines of code of the variant with periodic scheduling.

Infrastructure Size	Violation time (s)		No. migrations	
	period 30	reactive	period 30	reactive
128 nodes	20.92 ± 13.02	10.24 ± 4.48	62	107
256 nodes	23.33 ± 14.47	10.02 ± 3.32	124	201
512 nodes	23.57 ± 20.19	10.62 ± 7.21	269	421

Table 6: Snooze periodic *vs.* Snooze reactive.

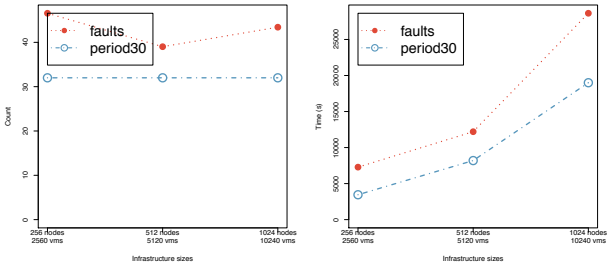
We have simulated reactive scheduling and a periodic algorithm for configurations ranging from 128 to 1024 LCs. In each case the Entropy scheduler has been applied to groups of 8 LCs per GM, the variant of hierarchical scheduling that performs most efficiently and closely matches the efficiency of DVMS, see Sec. 6.2.1. These simulations have yielded the results shown in Table 6 and in Figure 7 for the largest test (Figure 7 being an output of the trace analysis module of VMPlaceS). They clearly show that, while a reactive strategy entails a much higher number of migrations (because the periodic one aggregates overload situations and misses some of them), reactive scheduling results in a significantly lower total violation time.

6.2.3 *Analysis of Fault Tolerance Property*

We have performed a series of analysis of fault tolerance properties. For space reasons, we only detail below some corresponding results for the hierachical placement strategy. It turns out that Snooze's strategy based on heartbeats enables the reconstruction of the hierarchy in a relative short time and thus crashes on service nodes only moderately impact the resolution of violations even in the case of high failure rates.

As to Entropy, although the loss of the service node can be critical, its failure probability is so small that the single point of failure issue can be easily solved by a fail-over approach. Regarding DVMS, the crash of one node does not have any impact on the resolution as the composition of the microcosms is reevaluated immediately.

Regarding the hierarchical scheduling a la Snooze, we have also harnessed VMPlaceS in order to analyze and present to the best of our knowledge, a first analysis of such property. We have, in particular, compared standard executions (no faults, 32 LCs per GM, periodic calls every 30s to the centralized solver) to executions that have been subject to faults. In order be able to clearly evaluate the influence of faults, we have decided to analyze faulty executions under heavy stress: for a simulated time frame of 1800s, the following numbers of GM faults have been simulated: 10 GM faults for 256 LCs/8 GMs, 20 GM faults for 512 LCs/16 GMs, and 40 GM faults for 1024 LCs/32 GMs, that is, failure rates much higher than in real datacenters.



(a) GM Size (#LCs) (b) Cumulated Violation Time

Figure 8: Behavior in the presence of faults

Figure 8 shows two principal metrics for the standard (non-faulty) and faulty executions: the average GM sizes (in terms of LCs) during reconfigurations (left) and the violation times (right).

The results show that the average GM sizes are moderately affected by faults: 32 LCs in the case of non-faulty executions, 39–46 LCs in the case of the faulty executions. However, the violation times are rather strongly affected: 3453–18998s for the non-faulty executions, 7284–28636s in the presence of faults. This is very probably the case because the faults disturb on-going and future resolution actions triggered that are triggered by violations.

7. RELATED WORK

Jobs/tasks scheduling in distributed system is an old challenge and thus several simulators have been proposed to investigate pros and cons of new strategies for several years. As a recent example, Google has released the simulator [4] it used for the Omega framework [28]. However, jobs/tasks scheduling simulators do not consider the notion of VM and its associated capabilities (suspend/resume, migrate) and thus are not appropriate to investigate CC production platforms.

Simulator toolkits that have bee proposed to address CC concerns [12, 13, 21, 22, 26] can be classified into two categories: The first one corresponds to ad-hoc simulators that have been developped to address a particular concern. For instance, CReST [13] is a discrete event simulation toolkit built to evaluate Cloud provisioning algorithms. If ad-hoc simulators enable to provide some trends regarding the bevahiours of the system, they do consider the implication of the different layers, which can lead to non representative results at the end. Moreover, such ad-hoc solutions are developed for one shot and thus, they are not available for the scientific community. The second category [12, 22, 26] corresponds to more generic cloud simulator toolkits (*i.e.*, they have been designed to adress a majority of CC challenges). However, they have focused mainly on the API and not on the model of the different mechanisms of CC systems.

For instance, CloudSim [12], which has been widely used to validate algorithms and applications in different scientific publications, is based on a relatively top-down viewpoint of cloud environments. That is, there is no papers that properly validate the different models it relies on: a migration time is calculated by dividing a VM memory size by a network bandwidth. In addition to having inaccuracy weaknesses at the low level, available cloud simulator toolkits over simplified the model for the virtualization technologies, leading also to non representation results at the end. As highlighted several times throughout this document, we chose to build VMPlaceS on top of SimGrid in order to benefit fromt its accuracy of its models related to virtualization abstractions [19].

8. CONCLUSION

In this paper we have illustrated the use of VMPlaceS, a framework providing programming support for the definition of VM placement algorithms, execution support for their simulation at large scales, as well as new means for their trace-based analysis. VMPlaceS enables, in particular, the investigation of placement algorithms in the context of numerous and diverse real-world scenarios. This paper completes our previous introduction of VMPlaceS [23] and illustrates its relevance by discussing several experiments for three different classes of virtualization environments: centralized, hierarchical and fully distributed placement algorithms. We have also shown how VMPlaceS facilitates the implementation and evaluation of variants of placement algorithms. The corresponding experiments have provided the first systematic results comparing these algorithms in environments including up to one thousand of nodes and ten thousands of VMs in most cases.

The current version of VMPlaceS is available on a public git repository. [5] We are in touch with the SimGrid core developers in order to improve our code with the ultimate objective of addressing infrastructures up to 100K PMs and 1 Millions VMs over a period of one day.

As future work, we plan to add additional dimensions in order to simulate other workload variations stemming from network and HDD I/O changes. We are also investigating how it can be possible to replay real traces either from the Google cluster data set [6] or the more recent one provided by Bitbrains. [7] Finally, we are studying the current energy models that are present in SimGrid in order to be able to leverage these models to evaluate the energy footprint of consolidation strategies.

9. ACKNOWLEDGMENT

This work is supported by the French ANR project SONGS (11-INFRA-13). Experiments have been performed using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

10. REFERENCES

[1] CloudStack, Open Source Cloud Computing. http://cloudstack.apache.org.

[2] Open Source Data Center Virtualization. http://www.opennebula.org.

[3] PajeNG - Trace Visualization Tool. https://github.com/schnorr/pajeng.

[4] Simgrid publications. http://simgrid.gforge.inria.fr/Publications.html.

[5] Snooze web site. http://snooze.inria.fr, Last access: 21 Oct. 2014.

[6] The Open Source, Open Standards Cloud. http://www.openstack.org.

[7] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[8] A. Barker, B. Varghese, J. S. Ward, and I. Sommerville. Academic Cloud Computing Research: Five Pitfalls and Five Opportunities. In *Proceedings of the 6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, June 2014.

[9] L. A. Barroso and U. HÃűlzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 2009.

[10] R. Birke, L. Chen, and E. Smirni. Multi-resource characterization and their (in)dependencies in production datacenters. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–6, May 2014.

[4] https://github.com/google/cluster-scheduler-simulator

[5] http://beyondtheclouds.github.io/VMPlaceS/

[6] https://code.google.com/p/googleclusterdata/

[7] http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains

[11] V. A. Bloomfield. *Using R for Numerical Analysis in Science and Engineering*. Chapman & Hall/CRC, 2014.

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[13] J. Cartlidge and D. Cliff. Comparison of cloud middleware protocols and subscription network topologies using crest, the cloud research simulation toolkit - the three truths of cloud computing are: Hardware fails, software has bugs, and people make mistakes. In *CLOSER 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science, Aachen, Germany, 8-10 May, 2013*, pages 58–68, 2013.

[14] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, June 2014.

[15] L. Eyraud-Dubois and H. Larcheveque. Optimizing resource allocation while handling sla violations in cloud computing platforms. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 79–87, May 2013.

[16] E. Feller, L. Rilling, and C. Morin. Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds. In *CCGRID '12: 12th Int. Symp. on Cluster, Cloud and Grid Comp.*, pages 482–489, May 2012.

[17] F. Hermenier, S. Demassey, and X. Lorca. Bin Repacking Scheduling in Virtualized Datacenters. In *CP '11: Proceedings of the 17th international conference on Principles and practice of constraint programming*, pages 27–41. Springer, 2011.

[18] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: A consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pages 41–50, New York, NY, USA, 2009. ACM.

[19] T. Hirofuchi, A. Lebre, and L. Pouilloux. Adding a live migration model into simgrid: One more step toward the simulation of infrastructure-as-a-service concerns. In *Proceedings of the 2013 IEEE International Conference on Cloud Computing Technology and Science - Volume 01*, CLOUDCOM '13, pages 96–103, Washington, DC, USA, 2013. IEEE Computer Society.

[20] J. Hu, J. Gu, G. Sun, and T. Zhao. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*, pages 89–96, Dec 2010.

[21] A. Iosup, O. Sonmez, and D. Epema. Dgsim: Comparing grid resource management architectures through trace-based simulation. In E. Luque, T. Margalef, and D. Benítez, editors, *Euro-Par 2008 – Parallel Processing*, volume 5168 of *Lecture Notes in Computer Science*, pages 13–25. Springer Berlin Heidelberg, 2008.

[22] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5, Dec 2010.

[23] A. Lebre, J. Pastor, and M. SÃijdholt. VMplaceS: A Generic Tool to Investigate and Compare VM Placement Algorithms. In *To appear in Euro-Par 2015 Parallel Processing*, Lecture Notes in Computer Science. Springer, August 2015. Preprint available at http://people.rennes.inria.fr/Adrien.Lebre/PREPRINT/Europar-VMPlaceS.pdf.

[24] J. L. Lucas-Simarro, R. Moreno-Vozmediano, F. Desprez, and J. Rouzaud-Cornabas. Image transfer and storage cost aware brokering strategies for multiple clouds. In *IEEE CLOUD 2014. 7th IEEE International Conference on Cloud Computing*, Anchorage, USA, June 27-July 2 2014. IEEE Computer Society.

[25] R. Moreno-Vozmediano, R. Montero, and I. Llorente. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer*, 45(12):65–72, 2012.

[26] A. Nunez, J. Vazquez-Poletti, A. Caminero, G. Castané, J. Carretero, and I. Llorente. icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.

[27] F. Quesnel, A. Lebre, and M. Südholt. Cooperative and Reactive Scheduling in Large-Scale Virtualized Platforms with DVMS. *Concurrency and Computation: Practice and Experience*, 25(12):1643–1655, Aug. 2013.

[28] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 351–364. ACM, 2013.

[29] A. Shen, V. van Beek, and A. Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *IEEE/ACM CCGRID 2015*, May 2015.

[30] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[31] S. D. Team. Snooze characteristics and implementation, July 2014. Personal communication.

[32] H. N. Van, F. Tran, and J.-M. Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Computer and Information Technology, 2009. CIT '09. Ninth IEEE International Conference on*, volume 1, pages 357–362, Oct 2009.

[33] M. Wang, X. Meng, and L. Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 71–75, April 2011.