Scheduling Live-Migrations for Fast, Adaptable and Energy-Efficient Relocation Operations

Vincent Kherbache, Eric Madelaine, Fabien Hermenier

Scheduling Live-Migrations for Fast, Adaptable and Energy-Efficient Relocation Operations

Vincent Kherbache
INRIA Sophia Antipolis
Email: vincent.kherbache@inria.fr

Éric Madelaine
INRIA Sophia Antipolis
Email: eric.madelaine@inria.fr

Fabien Hermenier
University Nice Sophia Antipolis,
CNRS, I3S, UMR 7271
Email: fabien.hermenier@unice.fr

***Abstract*—Every day, numerous VMs are migrated inside a datacenter to balance the load, save energy or prepare production servers for maintenance. Despite VM placement problems are carefully studied, the underlying migration scheduler rely on vague *adhoc* models. This leads to unnecessarily long and energy-intensive migrations.**

We present mVM, a new and extensible migration scheduler. mVM takes into account the VM memory workload and the network topology to estimate precisely the migration duration and take wiser scheduling decisions. mVM is implemented as a plugin of BtrPlace and can be customized with additional scheduling constraints to finely control the migrations. Experiments on a real testbed show mVM outperforms schedulers that cap the migration parallelism by a constant to reduce the completion time. Besides an optimal capping, mVM reduces the migration duration by 20.4% on average and the completion time by 28.1%. In a maintenance operation involving 96 VMs to migrate between 72 servers, mVM saves 21.5% Joules against BtrPlace. Finally, its current library of 6 constraints allows administrators to address temporal and energy concerns, for example to adapt the schedule and fit a power budget.

***Keywords*-live-migrations; scheduling; energy-efficiency;**

I. Introduction

Infrastructure As A Service (IaaS) clouds provide clients with resources via Virtual Machines (VMs). To deploy applications (web services, data analytics *etc.*) in an IaaS cloud, a client installs the appropriate application and selects a Service Level Agreement (SLA) offered by the provider. Currently, public cloud providers advertise 99.95% availability [1], [2]. To ensure this, any management operation on the provider side must be done on the fly, with a minimal interference over the VM availability. Live migration [3] makes these management operations possible: it relocates a running VM from one server to another with a negligible downtime under idyllic conditions.

Today, live-migrations occur continuously. For example, dynamic VM placement algorithms relocate the VMs depending on their resource usage to distribute the load between the servers or to reduce the datacenter power consumption [4], [5], [6], [7]. These solutions work in two passes. The first pass consists in computing the new placement for some VMs according to specific objectives. The second pass consists in enacting the new VM placement using live-migrations. Datacenter operators heavily rely on live-migration to perform maintenance operations over production infrastructures [8]. For example, the VMs running on a server to update must be first relocated elsewhere to keep VMs availability. A maintenance operation occurs at the server scale but also at rack or cluster scale [9]. At a small scale, the operator may want to find a destination server and relocate the VMs by himself. At a larger scale, the operator is assisted by a placement algorithm.

A live-migration is a costly operation. It consumes network bandwidth and energy. It also temporarily reduces the VM availability. When numerous VMs must be migrated, it is important to schedule the migrations wisely, in order to minimize the impact on both the infrastructure and the delivered quality of service [10]. In practice, the duration of a migration depends on the allocated bandwidth and its memory workload. A sequential execution leads to fast individual migrations but long standing completion time. On the opposite, an excessive parallelism leads to a low per-migration bandwidth allocation hence long or even endless migrations. Additionaly, the datacenter operator and the customers have restrictions in terms of scheduling capabilities. For example, it may be required to synchronize the migration of strongly communicating VMs [11], while a datacenter must also cap its power usage to fit the availability of renewable energies or ensure power cooling capabilities [12]. This advocates for a scheduling algorithm that can take the benefits from the knowledge of the network topology, the VM workload but also the clients and the datacenter operator expectations to compute fast and efficient schedules.

Despite VM placement problems are carefully studied, we observe that the scheduling algorithms enacting the new placements do not receive the same level of attention. Indeed, underlying scheduling models that estimate the migration duration are often inaccurate. For example, Entropy [4] supposes a non-blocking homogeneous network coupled with a null dirty page rate. These hypotheses are unrealistic, prevent from computing efficient schedules and finally reduce the practical benefits of the placement algorithms [7].

In this paper, we present mVM, a migration scheduler that relies on realistic migration and network models to compute the best moment to start each migration and the amount of bandwidth to allocate. It also decides which migrations are executed in parallel to provide fast migrations and short completion times. In practice, mVM is implemented as a set of extensions for the customizable VM manager BtrPlace [13].

The evaluation of mVM is performed over a blocking network testbed against two representative schedulers: A unmodified BtrPlace that maximizes the migration parallelism similarly to [4], [6], [14], and a scheduler that reproduces Memory Buddies [15] decisions by capping the parallelism to a constant defined by the datacenter operator.

Our main results are:

Migration speed: On 50 migration plans generated randomly, the migrations scheduled by mVM completed on average 20.4% faster than Memory Buddies, while completion times are reduced by 28.1%. Contrarily to Memory Buddies, mVM always outperforms sequential scheduling with an average migration slowdown of 7.35% only, 4.5 times lower than with Memory Buddies.

Energy efficiency: In a server decommissioning operation involving 96 migrations among 72 servers, the schedule computed by mVM saves 21.5% Joules with regards to BtrPlace.

Scalability: Experiments show that mVM requires only 1.5 additional seconds with regards to BtrPlace to compute the schedule of a decommissioning operation involving 960 migrations among 720 servers.

Extensibility: mVM controls the scheduling at the action level through independent high-level constraints. The current library implements 4 constraints and 2 objectives. They address temporal and energy concerns such as the capability to compute a schedule fitting a power budget.

The paper is organized as follows. Section II describes the design of mVM. Section III details its implementation and Section IV presents performance optimizations. Section V evaluates mVM. Finally, Section VI describes related work, and Section VII presents our conclusions and future works.

II. mVM OVERVIEW

mVM is a migration scheduler that can be configured with specific constraints and objectives. It aims at computing the best sequence of migrations along with any actions needed to perform a data center reconfiguration while continuously satisfying the constraints. It is implemented as a set of extensions for BtrPlace and controls VMs running on top of the KVM virtual machine monitor [16]. In this paper, we refer to a customized version of BtrPlace with our extensions as mVM.

In this section, we first introduce the architecture of mVM and illustrate how it concretely performs migration scheduling.

A. Global design

Figure 1 depicts the architecture of mVM. mVM takes as input three types of informations, the data center configuration, the VM characteristics and the scheduling constraints. The datacenter configuration specifies the network including its topology along with the capacity and the connectivity of the switches. This information is usually obtained automatically by a monitoring tool. Despite mVM should be able to comply with any tool, these informations must be provided using the SimGrid Platform Description Format.[1]

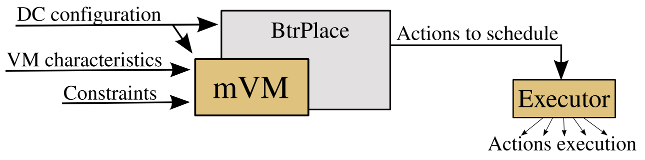

Figure 1. mVM architecture.

The VM characteristics provides the current VM placement and resource usage but also their real memory usage and their dirty page rate. All these informations can also be retrieved by a monitoring tool. The memory usage and dirty page rate are however rarely monitored. We then develop a command to retrieve these informations from *libvirt*.

The constraints indicate the expectations that must be satisfied by the computed schedule. They must at least state the future hosting server for each VM. These constraints can be specified manually or computed with a VM placement algorithm; With the legacy version of BtrPlace for example. The constraints also express additional restrictions such as the need to synchronize some migrations or to cap the datacenter power usage during a reconfiguration. They can be provided through configuration scripts or directly through an API.

With these inputs, mVM computes a *reconfiguration plan* that is a schedule of actions to execute. For each migration action, mVM indicates the moment to start the action, its predicted duration and the amount of bandwidth to allocate.

The *Executor* module applies the schedule by performing all of the referred actions. In practice, it is not safe to execute actions by only focusing on the predicted start times as the effective duration of an action may differ from its estimated duration. This can lead to unexpected SLA violations, an extra energy consumption, or a technical limitation such as the migration of a VM to an server that is not yet online. To address this issue, the executor inserts dependencies between actions according to a global virtual clock.

B. Scheduling example

Figure 2 illustrates a network topology that connect 6 servers grouped into two racks by a single 2 Gbit/s link. The VMs from the source servers must be migrated to the destination servers using a given placement. The length of a VM represents its duration when migrated at 1 Gbit/s.

The Gantt chart in Figure 3 depicts a possible migration schedule. According to the network topology, the bottleneck

[1]http://simgrid.gforge.inria.fr/simgrid/3.9/doc/platform.html

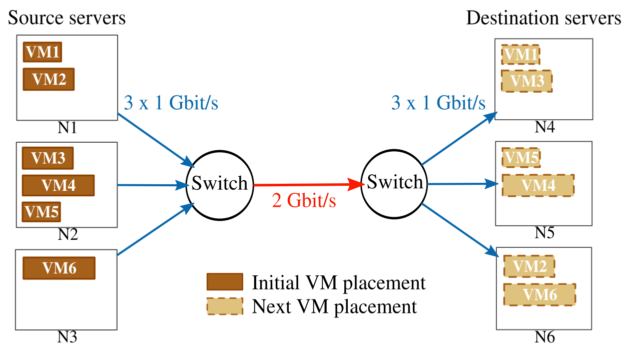

Figure 2. Sample migration scenario.

is the 2 Gbit/s inter-switch link. It is however possible to allocate 1 Gbit/s to each migration and schedule them 2 by 2 from different source and destination servers, to fully exploit the inter-switch link. This sample schedule takes then the full advantage of both inter-server parallelization and intra-server serialization to continuously use the inter-switch link at its maximum bandwidth and provide fast migrations.

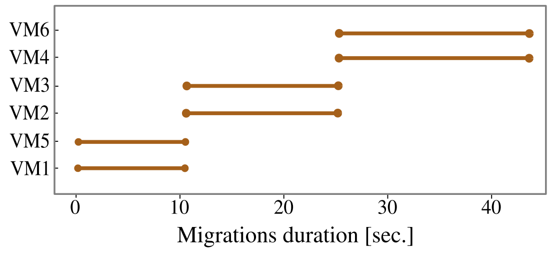

Figure 3. A schedule for the scenario in Figure 2.

III. Implementation

In this section, we describe the implementation of mVM. We first introduce BtrPlace architecture. We then provide details of the implementation of the network and the migration model. We finally present extensions we developed on top of the migration model to control the scheduling with regards to temporal or energy-efficiency concerns.

A. *BtrPlace architecture*

BtrPlace [13] aims at computing the next placement for the VMs, the next state for the servers, and the action schedule that lead to this stage.

BtrPlace uses constraint programming (CP) to model a placement for the VMs and the action schedule, it relies on the Java library Choco [17] to solve the associated problem. CP is an approach to model and solve combinatorial problems in which a problem is modeled by logical relations that must be satisfied by the solution. The CP solving algorithm is independent of the constraints composing the problem and the order in which they are provided. To use CP, a problem is modeled as a *Constraint Satisfaction Problem* (CSP), comprising a set of *variables*, a set of *domains* representing the possible values for each variable, and a set of *constraints* that represent the required relations between the values and the variables. A solver computes a solution for a CSP by assigning each variable to a value that simultaneously satisfies the constraints. The CSP can be augmented with an objective represented by a variable that must have its associated value maximized or minimized. To minimize (resp. maximize) a variable K, Choco works incrementally: each time a solution with an associated cost k is computed, Choco automatically adds the constraint $K < k$ (resp. $K > k$) and tries to compute a new solution. This added constraint ensures the next solution will have a better objective value. This process is repeated until Choco browses the whole search space or hits a given timeout. It then returns the last computed solution.

From its inputs, Btrplace first models a core *Reconfiguration Problem* (RP), *i.e.* a minimal placement and scheduling algorithm that manipulate servers and VMs through actions. Each action is modeled depending on its nature (booting, migrating or halting a VM, booting or halting a server). An action $a \in \mathscr{A}$ embeds at least a variable $st(a)$ and $ed(a)$ that denote the moment the action starts and terminates, respectively.

As CP provides composition, it is possible to plug external models on top of the core RP to support additional datacenter elements, such as the network, but also additional concerns such as the power usage that results from the execution of each action. It is also possible to use an alternative model for each kind of action. Once the core RP is generated, BtrPlace customizes it with all the stated constraints and the possible objective. The resulting specialized RP is then solved to generate the action schedule to apply.

mVM inserts inside the core RP a network model, a new migration model and a power model to formulate the power consumption of a migration. On top of the core RP, mVM also provides additional constraints and objectives to adapt the schedule with regards to temporal and energy concerns. In total, these extensions represent 1600 lines of Java code.

B. *Network model*

A migration transfers a VM from a server to another through a network. For economic and technical reasons, a network is rarely non-blocking. Indeed, network links and switches might not be provisioned enough to support all the traffic in the worst case scenario.

Our network model represents the traffic generated by each migration over the time and the available bandwidth, through a set of network elements. All the links are considered full-duplex. As the next VM placement is known, the model considers that a VM migrate from its source to its destination server through a predefined route. The bandwidth allocation for a migration is also supposed to be constant. Finally, the model ignores the network latency, which means that it considers a migration occupies simultaneously all the networking elements it is going through. This assumption is coherent as temporal variables in our model are expressed in terms of seconds while the network latency between two servers in a datacenter is much less than a second.

The network model considers a set of VM migrations $\mathscr{M} \subseteq \mathscr{A}$ to perform over a set of network elements $\mathscr{N}$ (network interfaces, switches, *etc.*). For any element $n \in \mathscr{N}$, $capa(n)$ denotes its capacity in Mbit/s. For any migration $m \in \mathscr{M}$, $path(m) \subseteq \mathscr{N}$ indicates the network elements crossed (source and destination servers included), $bw(m)$ denotes the allocated bandwidth, $st(m)$ and $ed(m)$ indicate the beginning and the end of the operation, respectively. The equation (1) models the bandwidth sharing of a network element among the migrations that pass through it:

$$\forall n \in \mathscr{N}, \forall t \in \mathbb{N}, \quad \sum_{\substack{m \in \mathscr{M},\ n \in path(m), \\ t \in [st(m);\ ed(m)]}} bw(m) \quad < \quad capa(n) \qquad (1)$$

In practice, the bandwidth sharing is modeled with cumulative constraints [18]. A cumulative constraint consists in placing a set of tasks on a bounded resource. Each task has three variables: a height, a duration, and a starting time. The constraint ensures then that at any time, the cumulative height of the placed tasks does not exceed the height of the resource. We use one cumulative constraint per network element where each task represents a migration and the height corresponds to the available bandwidth.

C. Migration model

The migration model mimics the pre-copy algorithm [3] used in Xen and KVM. The model assumes a shared storage for the VM disk images. The pre-copy algorithm is an iterative process. The first phase consists in sending all the memory used by the VM to the destination server while the VM is still running. The subsequent phases consist in sending iteratively the memory pages that were made dirty during the previous transfer. Thus, the migration duration depends of the memory dirtying rate and the bandwidth allocated to the migration. The migration terminates when the amount of dirty pages is sufficiently low to be sent in a time interval lesser than the *downtime* (30 ms by default). Once this condition is met, the VM is suspended on the hosting server, the latest memory pages are transferred, and the VM is resumed on the destination server. It is worth noting that with this algorithm, the duration of a live-migration increases exponentially when the allocated bandwidth decreases linearly (see Figure 4).

According to the majority of the loads observed on real applications [19], [20], the observation of the memory dirtying rate can be separated in two phases. The first phase corresponds to the *hot-pages*, a set of memory pages that are quickly dirtied. This phase exhibits a high dirty page rate but

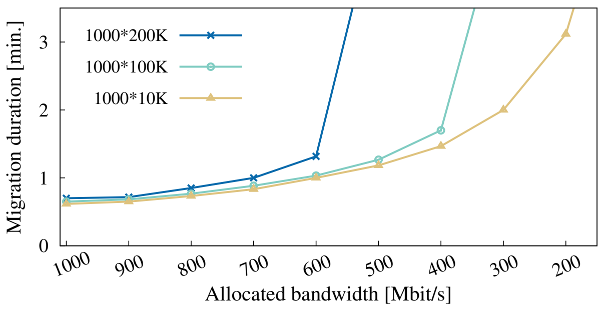

Figure 4. Duration of a live-migration between 2 KVM hypervisors depending on the allocated bandwidth and the parameters used by the command *stress* to generate dirty pages. *1000*10K* indicates the VM runs 1000 concurrent threads that continuously allocate and free up 10 KiB of memory each. The VM memory used is set to 4 GiB and the downtime is limited to 30 ms.

during a short period of time. The second phase represents the linear evolution of the *cold-pages* which corresponds to the pages that became dirty after the generation of the *hot-pages* until the end of the migration. These two phases are distinguished by the observation of the memory dirtying rate variation. The amount of *hot-pages* HP_s in MiB, and the seconds HP_d spent to rewrite them determine the *hot-pages* rate $HP_r = \frac{HP_s}{HP_d}$, this rate gives a good overview of the minimum bandwidth to allocate to ensure the termination of a migration. In practice, predicting the termination of a migration consists in measuring HP_r over a period equal to the downtime period D and ensuring that its rate is less than the available bandwidth. Indeed, the memory pages marked as dirty in this time interval will be transferred in the final iteration of the pre-copy algorithm, thus, they must be transferred in a period less than D. CP_r corresponds to the *cold-pages* rate in MiB/s, it is measured from $t = HP_d$. Often very low, this rate is still dependent of the VM's workload.

Given a migration $m \in \mathscr{M}$, with $mu(m)$ the amount of memory used by the VM in MiB and $bw(m)$ its allocated bandwidth, the minimum duration of the migration d_{min} is written: $d_{min}(m) = \frac{mu(m)}{bw(m)}$. Generally, to transfer an amount of memory X with a bandwidth Y and a memory dirty rate Z, the transfer duration is: $\frac{X}{Y-Z}$.

The cold pages are rewritten at the beginning of the migration process just after the hot pages. Hence if we assume that d_{min} is always higher than the time required to dirty the hot pages, the total amount of cold pages CP_s corresponds to: $CP_s = \left(d_{min}(m) - HP_d\right) \times CP_r$. Thus the time spent to send the cold pages d_{CP} is written:

$$d_{CP}(m) = \frac{HP_s + CP_s}{bw(m) - CP_r} \qquad (2)$$

Then the time spent to send the hot-pages d_{HP} equals:

$$d_{HP}(m) = \frac{HP_s}{bw(m)} \times \frac{HP_r}{bw(m) - HP_r} \qquad (3)$$

Finally, the duration d of a migration m is:

$$d(m) = d_{min}(m) + d_{CP}(m) + d_{HP}(m) + D \qquad (4)$$

The duration of d_{min} is the dominating factor. It is usually expressed in terms of seconds or minutes, while $d_{CP}(m)$ and $d_{HP}(m)$ are usually expressed in seconds. Finally the downtime D has a very low weight (30 ms by default). It can thus be ignored when the unit of time is a second.

This migration model establishes the link between the duration of a migration, represented by the length of the task in a cumulative constraint, and the bandwidth to allocate, represented by the height of the task. As a result, mVM knows that a minimum bandwidth is required to ensure the migration termination while allocating a high bandwidth reduces the migration duration exponentially.

D. Extensions

In this section we present the extensions we developed to control the migrations. All these extensions were implemented using the original BtrPlace API and rely on the variables provided by the migration model.

1) Temporal control: Sync synchronizes the migrations of the VMs passed as parameters. It is a constraint inspired by COMMA [11]. When two strongly-communicating VMs must be migrated to a distant server, they can be migrated sequentially. Temporarily, one VM will be then active on the distant server while the second one stay on the source server. The two VMs will thus suffer from a performance loss due to a communication through a slow link. It is possible to migrate a VM using either a pre-copy or a post-copy algorithm [21]. While in the pre-copy algorithm, the VM state is migrated at the end of the operation, the post-copy algorithm migrates the VM state at the beginning of the operation. *Sync* supports both approaches and can synchronize either the beginning or the end of the migrations. In practice, the constraint enforces the variables denoting the moment the migrations starts (post-copy algorithm) or end (pre-copy algorithm) to be equal.

Before establishes a precedence rule between two migrations or a migration and a deadline. It allows a datacenter operator to specify priorities in a maintenance operation for example, or to ensure the termination of a heavy maintenance operation in time, before the office hours for example. The constraint that establishes a precedence rule between two migrations m_1 and m_2 is expressed as follows:

$$ed(m_1) \leq st(m_2)$$

Seq ensures that the given migrations will be executed sequentially but with no precise order. This allows the operator to reduce the consequences if a hardware failure occurs during the execution of a schedule as only one migration will be active. The constraint does not force any ordering to let the scheduler decides the most profitable one with respect to the other stated constraints. *seq* is implemented by a cumulative constraint with a resource having a capacity of 1 and each migration a height of 1. An implementation based on a *disjunctive* [22] constraint would be preferable to obtain better performance. It is however not yet implemented inside Choco.

MinMTTR is an objective that ask for fast schedules. The intuition is to have fast actions that are executed as soon as possible. It is implemented as follows:

$$\min\left(\sum_{a\in\mathscr{A}} ed(a)\right)$$

2) Energy aware scheduling: A schedule is composed of some actions to execute. In a server maintenance operation for example, there will be VMs to migrate but also servers to turn on or off. These operations should be planned with care to consume a few amount of energy or a consumption that fit a given power budget [12]. BtrPlace already embeds a power model for the actions that consists in turning on and off a server or a VM. We describe here the power model for a migration and two constraints to control the energy usage during a reconfiguration.

The energy model derives from the model proposed by Liu *et al.* [19]. The amount of data transmitted and received by these servers is the same. With network interfaces that are not energy adaptive, the authors propose and validate a model where the energy consumed by a migration increases linearly with the amount of data to be transferred. Equation (5) formulates with variables of our migration model, the energy consumption of a migration when the source and the destination servers are identical. α and β are parameters that must be computed during a training phase.

$$\forall m \in \mathscr{M}, E(m) = \alpha \times bw(m) \times d(m) + \beta \qquad (5)$$

PowerBudget controls the instantaneous power consumed by the infrastructure during the reconfiguration process. It takes as parameters a period of time and the power capping. This constraint is required for example to avoid overheating [12], or when the datacenter is powered by renewable energies or under the control of a Smart City authority that restricts its power usage. Using *PowerBudget*, mVM can then delay some migrations or any actions, depending on their power usage. *PowerBudget* is implemented using a *cumulative* constraint. The resource capacity is the maximum power allowed during the reconfiguration. Each action is modeled as a task with its height denoting its power usage. Finally, when the power budget is not a constant for the whole duration of the reconfiguration process, additional tasks are inserted to create a power profile aligned with the requirements.

MinEnergy is an objective that minimizes the overall energy consumed by the datacenter during the reconfiguration.

The cost variable to minimize is defined as the sum of the energy spend by each action. It is implemented as follows:

$$\min\left(\sum_{a\in\mathscr{A}} E(a)\right)$$

The overall implementation is succint, each constraint reprensents approximately 100 lines of Java code, while each objective requires around 200 lines.

IV. Optimizing mVM

The problem of computing for each migration a time to start the action and a bandwidth to allocate refers to the *Resource-Constrained Project Scheduling Problem* where each migration is a task and each network element is a resource to share. This problem is known to be NP-hard. Therefore, computing a solution is time consuming when the number of VMs is large. mVM uses two strategies to optimize the solving process. Our first strategy simplifies the problem using a domain specific hypothesis while the second is a heuristic that guides the CP solver toward fast migration plans.

The *MaxBandwidth* optimization precomputes the bandwidth to allocate to the migrations. As stated in Section III-B, there is a limited interest in parallelizing migrations up to the point of sharing the minimal bandwidth available on the migration path: this increases the amount of memory pages to re-transfer and thus the migration duration. Accordingly, this optimization forces to allocate the maximum bandwidth for each migration. As a side effect, this simplification precomputes the migration duration as well. *MaxBandwidth* reduces then the set of variables in the problem to the variables denoting the moment to start the migrations. Despite this simplification, the problem stays NP-hard.

Our second strategy is a domain-specific heuristic that indicates to the solver the variables it has to instantiate first and the values to try for these variables. In general, the intuition is to guide the solver to variable instantiations of interest. First, the heuristic establishes different group of variables. Each group contains the start moment of the possible action to perform over the server, the start moment of each VM that must be migrated to that server and the possible actions to perform over the source servers. Second, the heuristic asks the solver to instantiate the start moments group by group. Until all the variables of a group are not instantiated, the heuristic asks to focus on the hardest action to schedule, *i.e.* action that has the smallest domain for its start variable. Once the variable is selected, the heuristic asks then to try to instantiate the variable to its smallest allowed value, so to start the action as soon as possible. It is worth noting that the heuristic is only a guide, it does not change the problem definition and still leads to a viable solution. Indeed, the solver prevents to perform an instantiation that contradicts a constraint and allows the backtracking mechanism to revise a initial instantiation that turned to be invalid later in the decision tree.

V. Evaluation

mVM aims at improving the live-migration scheduling thanks to an accurate migration model and appropriate decisions. In this section, we evaluate the practical benefits of mVM in terms of migration and reconfiguration speedup over a network testbed. We also validate the capability of mVM to address energy concerns and evaluate its scalability.

A. Testbed setup

All the experiments were conducted on the Grid'5000 platform [23]. The testbed is composed of racks hosting 24 servers each. Servers in a same rack are connected to a Top-of-Rack (ToR) switch through a Gigabit Ethernet interface. All the ToR switches are then connected together through a 10 Gigabit Ethernet aggregation switch. Servers are also connected to a 20 Gbit/s Infiniband network. For a better control of the network traffic, the VM disk images are shared by dedicated NFS servers through the Infiniband network while all the migrations transit through the Ethernet network. We consider a dedicated migration network to avoid any interference with the VM network traffic; a common practice in production environment [24]. Each server runs a Debian Jessie distribution with a GNU/Linux 3.16.0-4-amd64 kernel and the Qemu (KVM) hypervisor 2.2.50. The VM configuration and the migrations were performed using *libvirt*. Each VM runs a Ubuntu 14.10 desktop distribution with a single virtual CPU, the maximum migration downtime is 30 ms and the workloads are generated using *stress*.

B. mVM to speed up migrations

The experiment consists in scheduling and executing the migration of 10 VMs with different memory usage between 4 servers connected through an heterogeneous network. Each server has 2 quad-core Intel Xeon L5420, 16 GiB RAM and is connected to a central switch through a Gigabit Ethernet interface. To emulate a blocking network, the *tc* command limits the network bandwidth of two servers at 500 Mbit/s. The VM memory used is set to 2 and 3 GiB, equally distributed among the VMs. This amount represents the real memory allocated to the guest by *Qemu*, thus the one transferred during the migration. The memory workload for each VM is generated by 1,000 threads that continuously write 70 KiB of RAM.

In this experiment, we compare the schedules computed by mVM against a scheduler that reproduces Memory Buddies [15] decisions. Similarly to mVM, Memory Buddies controls the migration parallelism, however it limits the parallelism to a constant to be defined. In practice, we compare mVM to three configurations of Memory Buddies, referred as MB-2 to MB-4, where the parallelism varies from 2 to 4. To perform a robust experimentation that covers a wide spectrum of scenarios, we precomputed 50 runs of 10 migrations each where the initial and the destination server for each VM are computed randomly. Each run has been executed 3 times for each VM scheduler.

Table I
ABSOLUTE MIGRATION DURATIONS AND RELATIVE SLOWDOWN COMPARED TO A SEQUENTIAL SCHEDULING

Scheduler	mVM	MB-2	MB-3	MB-4
Mean migration time (sec.)	45.55	57.22	113.2	168.6
Mean slowdown (%)	7.35%	29.69%	141.3%	259.2%

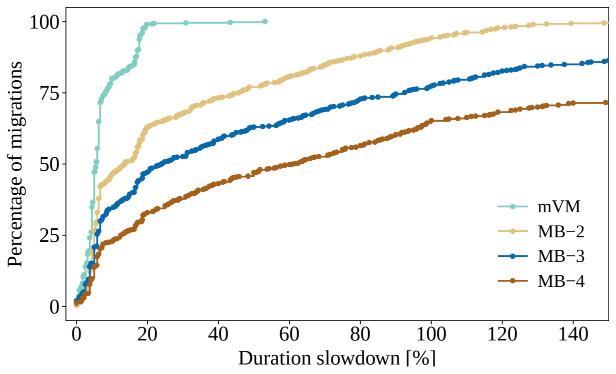

Figure 5. CDF of migrations duration slowdown compared to sequential predictions. Migrations with a slowdown greater than 150% are not displayed.

Table I summarizes the average migration duration for each scheduler. We first observe mVM outperforms every configuration of Memory Buddies. Indeed, the migrations scheduled by mVM completed 20.4% faster than those computed by MB-2, the best Memory Buddies configuration. To assess the absolute quality of these results, we compared the durations to sequential migrations computed on a flawless virtual environment. This exhibits the potential migration slowdowns due to parallelism decisions. We observe an average 7.35% slowdown for mVM while it is at least 4.5 times higher for MB-2. Figure 5 depicts the migration slowdowns as a CDF. We observe 88.8% of the migrations scheduled by mVM have a slowdown of 5 seconds at maximum, against at least 52.8% for MB-2. We also observe the slowdown distribution for mVM is gathered while it is scattered for Memory Buddies and increasing with the concurrency. As the 50 different migration plans were generated randomly, this shows mVM is more reliable than Memory Buddies to get fast migrations as the performance is not heavily dependent from the context.

These improvements over Memory Buddies are explained by better parallelism decisions. Indeed, Memory Buddies parallelizes the migrations statically without any knowledge about network topology or VM placement. This can produce an insufficient usage of the overall network capacity and an undesired concurrency between migrations on a same network path. This reduces the migration bandwidth, thus leads to more retransmissions of dirty memory pages and higher migration durations. On the other side, mVM infers the optimal number of concurrent migrations over the time from its knowledge of the network topology. In practice, we observed the number of concurrent migrations varied from 2 to 5. We also observe mVM took better parallelism decisions than the most aggressive Memory Buddies configuration while producing a lower slowdown than the most conservative one. As a result, mVM migrates each VM at maximum speed and parallelizes them to maximize the usage of the network capacity. We finally observe three abnormally long migrations with mVM. A post-mortem analysis reveals these durations were caused by the technical limitations of our testbed. Indeed, when a server sends and receives migrations simultaneously at maximum speed through a 500 Mbit/s limited interface then the traffic shaping queuing mechanism is not fair and we observe periodic bandwidths slowdown. We reproduced this disruption using the *iperf* tool and measured a slowdown varying from 100 Mbit/s to 200 Mbit/s. This problem also occurs using Memory Buddies as the chances to migrate multiple VMs on a same link increased with the parallelism.

Table II
ABSOLUTE COMPLETION TIMES AND RELATIVE SPEEDUP COMPARED TO A SEQUENTIAL SCHEDULING

Scheduler	mVM	MB-2	MB-3	MB-4
Mean completion time (sec.)	212.8	295.9	394.6	479.4
Mean speedup (%)	54.18%	36.42%	15.94%	-2.64%

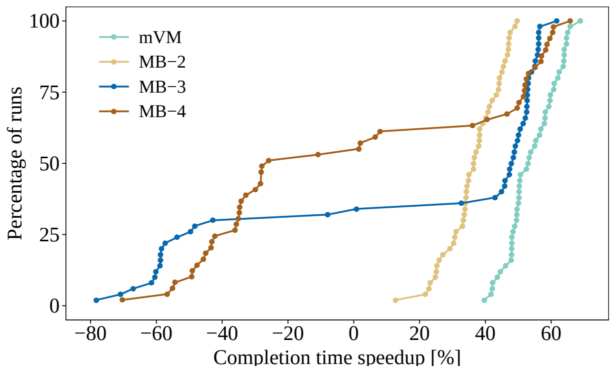

Figure 6. CDF of completion times speedup compared to sequential executions.

Table II shows mVM produces shorter completion times than Memory Buddies. We observe executions completed on average 28.1% faster than with MB-2, the best configuration for Memory Buddies. mVM completed the executions in average 83.1 seconds earlier than MB-2. To assess the quality of these results, we compared completion times to predicted values of a pure sequential scheduling. We observe an average speedup of 54.18% for mVM while it is at least

1.49 times lower with MB-2. Figure 6 depicts the completion times speedup as a CDF. It first confirms mVM exhibits the most important speedup. We also observe the speedups for MB-3 and MB-4 are scattered and not always positives. This confirms mVM offers a reliable and a performant scheduling algorithm.

This overall improvement is due to the parallelism and clustering decisions taken by mVM. As explained before, mVM optimizes the parallelism according to the migration routes and the available bandwidth while Memory Buddies decisions are capped by a constant. Furthermore, contrarily to Memory Buddies, mVM infers how to group the migrations according to their predicted duration. This reduces the periods where the network is underused and consequently the completion time. As a conclusion, this experiment confirmed that predicting the migration duration to compute an adaptive level of parallelism and a tight migration clustering is a key to compute efficient schedules. Indeed, while mVM computed the shortest plans, no particular configuration of Memory Buddies outperform the others.

A part of the experimental gain of mVM comes from decisions based on an analysis of the VM dirty page rate. Despite such an approach is a common practice in the state of the art and has already been tested under production workloads [10], [19], [20], some VMs might still have a fuzzy dirty page rate. In this case the estimated migration duration might be inaccurate then fool mVM. However, this does not prevent mVM to compute wise schedules with regards to Memory Buddies. Indeed, despite these mis-estimations might bias the clustering decisions thus extend the completion time, they have no impact on mVM parallelism decisions that solely depends on the network model. Unlike Memory Buddies there will still be no excessive parallelism decisions, therefore keeping migrations as short as possible.

C. mVM to address energy efficiency

This experiment evaluates the practical benefits of mVM when addressing energy concerns during migrations. It consists in executing a decommissioning scenario over multiple servers and observe the capabilities of mVM to compute schedules that consume less energy or to restrict the overall power consumption. Contrary to BtrPlace, Memory Buddies cannot schedule the actions that consists of turning on or off servers. Accordingly, we use the original BtrPlace as a representative baseline for this experiment.

The testbed is composed of 3 racks. Each rack consists of 24 servers with one Intel Xeon X3440 2.53 GHz CPU and 16 GiB RAM each. ToR switches connect the servers through a Gigabit Ethernet while the ToR switches are connected to a 10 GBit/s aggregation switch. The decommissioning scenario consists of migrating the VMs from two racks to the third one. To save power, the destination servers are initially turned off and the server to decommission have to be turned off once their VMs are migrated. Each source server hosts 2 VMs. This amounts to 96 VMs to migrate from 48 to 24 servers. Every VM uses 1 virtual CPU and the allocated memory is set to 2 GiB and 4 GiB RAM equally distributed among the VMs.

To calibrate the energy models with realistic values, we reused the experimental values from [19] for the migration energy model while the idle energy consumption of the servers were measured directly from the testbed (see Table III).

Table III
ENERGY MODEL CALIBRATION

Model element	**Energy model**
Server consumption (*idle*)	110 $W\times$ *running duration*
Server boot overhead	20 $W\times$ *boot duration*
VM hosting	16%$\times$ *idle* $\times$ *hosting duration*
Migration	0.512$\times$ *transferred data* +20.165

1) Energy saving capabilities: Figure 7 compares the power usage of the same decommissioning scenario scheduled by either BtrPlace or mVM. As the testbed is not instrumented enough to measure the power consumption of each server, the values were computed from the energy model. We observe that mVM saved a total of 2,350 Joules compared to BtrPlace, a 21.55% reduction. This is explained by the schedule computed by mVM that allowed to turn off the source servers sooner thanks to faster migrations. At the beginning of the experiment, the instantaneous power consumption grows up from 7 kW to 10 kW with both schedulers. This increase is explained by the simultaneous boot of the 24 destination servers during 2 minutes. Once available, BtrPlace launches all the migrations in parallel. This results in very long migration durations. As all the migrations terminate almost simultaneously at minute 7, it is then impossible to turn off any source server before that time. With mVM, migrations complete faster and some source servers are being turned off from minute 2. This behavior can be seen by the regular going down steps on Figure 7.

We observed mVM scheduled the migrations 10 by 10. These groups were defined to maximize the bandwidth usage and minimize the migration duration. As stated in Section IV, the *MaxBandwidth* optimization forces a 1 Gbit/s bandwidth per migration, so the 10 by 10 parallelization fully utilizes the 10 Gbit/s link that is connected to the destination switch. Also, in order to obtain a 10 Gbit/s data flow, the migration groups where all chosen from 10 different source and destination servers at a time and grouped by their predicted duration. With mVM, we also observe small peaks in the energy consumption. They correspond to the termination of a migration group and the beginning of a new one. In theory, these sequences follow on from each other. However the small predictions errors (around 7%) imply to synchronize these transitions to maintain the original

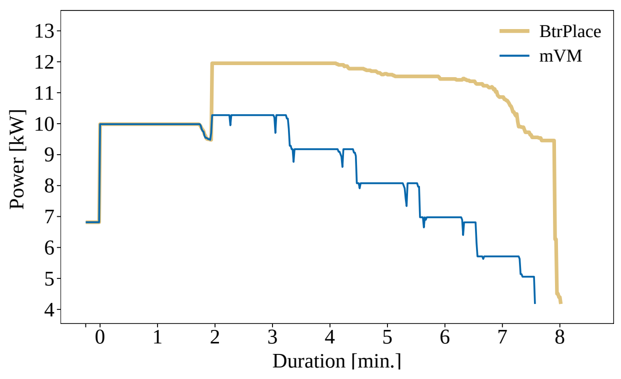

Figure 7. Energy consumption.

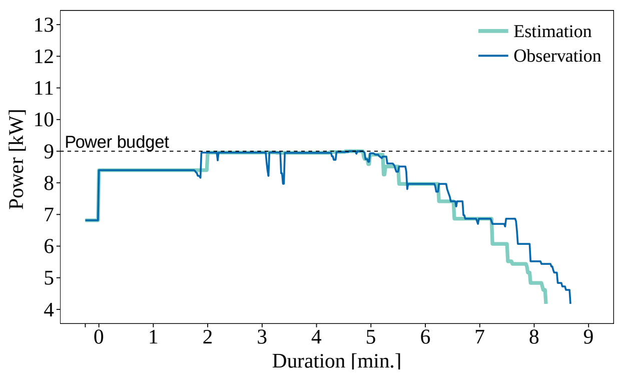

Figure 8. Impact of a power capping on the power usage.

schedule and avoid actions overlapping. At the end, the completion time exceeded the prediction by only 5 seconds.

2) Power capping capabilities: The *PowerBudget* constraint restricts the instantaneous power consumed by the infrastructure during the reconfiguration process. As it restricts the consumption over the time, this constraint can delay some migrations or any actions, depending on their power usage. To verify the effectiveness of the power budget constraint on the scheduling decisions, we executed the decommissioning scenario under a restrictive power budget of 9 kW.

Figure 8 shows the power consumption of the predicted and the observed scheduling. We first observe mVM reduced the peak power consumption to stay under the threshold. In practice, the *PowerBudget* constraint forced to spread the boot actions during the first 5 minutes of the execution. A first set of actions was executed at the beginning of the experiment to finish at minute 2. Then, the remaining actions where scheduled later, in smaller groups that partially overlap. From minute 2 to 5, we observe the power consumption is very close to the 9 kW budget. Indeed, mVM executed a few migrations in parallel to fill the gap and to try to terminate the operation as soon as possible. It was however not possible to migrate the VMs 10 by 10 contrary to the previous experiment. As a result, the operation required 1.5 additional minutes to complete with regards to an execution without *PowerBudget* (see Figure 7).

Despite we measured a prediction accuracy of 93% for the migration durations, we observe the practical completion time exceeds the predicted one by 32 seconds. This is mainly explained by the larger number of synchronization points inserted by the Executor to maintain the computed sequence of migrations and thus comply with the capping constraint. There is also an inevitable latency that is due to the time to contact the hypervisors, initiate the migrations and wait for KVM to reach the expected transfer rate.

D. Scalability

Computing the moment to start each migration with regards to bandwidth requirements is NP-Hard. In practice, the time required by mVM to compute a schedule depends on the amount of VMs to migrate, the number of network elements, and their bandwidth capacity. In this experiment, we evaluate the solving duration of mVM by computing schedules for a decommissioning scenario that is scaled up to 10 times with regards to the experiment performed in Section V-C. The scaling factor is applied on the aggregation switch capacity and the number of racks. At the largest scale, mVM and BtrPlace must then compute a schedule for a decommissioning scenario that requires to migrate 960 VMs running inside 20 racks of 24 servers each, to 10 new racks. While all the servers are still connected to their ToR switch through a Gigabit Ethernet link, the aggregation switch provides a 100 Gbit/s bandwidth which we consider as an exceptional bandwidth for a datacenter.

Figure 9 depicts the computation time that was needed by BtrPlace and mVM to compute their best solution and to prove their optimality. As expected, we first observe the solving duration increases exponentially due to the nature of the problem. We however observe this duration is low and the performance overhead of mVM compared to BtrPlace is acceptable with regards to its benefits in terms of migration speed and energy efficiency. At the largest scale, mVM requires only 1.5 additional seconds to compute the best possible schedule. For this evaluation, both the number of racks and the aggregation switch capacity are scaled linearly. We should then observe a completion time reduced by 30 seconds with regards to BtrPlace, similarly to the real experiment conducted at scale 1. This overhead is negligible with regards to the time that is needed to execute the resulting schedule.

At a very larger scale, the solving duration for mVM will become significant. A solution to overcome this limitation would be to part the operation in multiple steps. At the moment the bandwidth used to migrate VMs exceeds the

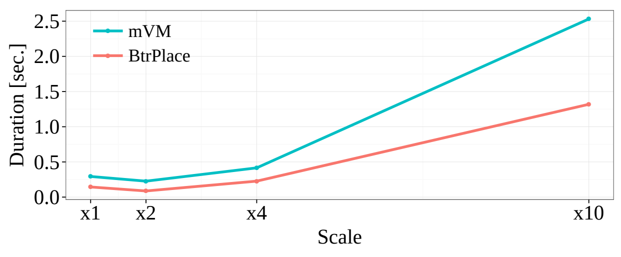

Figure 9. Solving duration of mVM and BtrPlace depending on the scale of the decommissioning operation.

aggregation switch capacity, mVM migrates the VMs by group. Accordingly, with a 100 Gbit/s interconnect, asking mVM once to migrate 960 VMs or asking mVM twice to migrate 480 VMs at each step would lead to the same observable result while being less stressful for the datacenter operator.

VI. Related works

Migration scheduling in VM managers: Dynamic VM placement algorithms embed schedulers to solve action dependencies and ensure their theoretical feasibility. For example, [4], [13], [15] can delay a VM migration to a server when the memory or CPU requirements are not yet available due to a pending outgoing migration on that server.

Current approaches are however incomplete when they want to control the scheduling to get fast migrations. Many works [4], [6], [13], [14] estimate the migration duration to be equal to the VM memory usage divided by the network bandwidth. The experiments discussed in Section V proved that this assumption is not realistic. This ignores the principles of the pre-copy algorithm or assumes that the VMs do not write into their memory. It also assumes a non-blocking network where none of the VMs to migrate are co-located. Memory buddies [15] discusses the impact of concurrent live-migrations. They propose to cap the concurrency with a number to be defined. The experiments discussed in Section V also proved that this assumption is not optimal. Indeed, this solution improves the practical quality of the scheduling in some cases but the concurrency cannot be constant as it depends on the current network load and the migration path. COMMA [11] considers the network bandwidth and the dirty page rate to synchronize in real time the termination of strongly communicating VM migrations. It however assumes a single network path for all the VMs. mVM implements the concept of COMMA with the *Sync* constraint. It however has no network limitation with the knowledge of the whole topology.

Predicting live-migration duration for simulation: The simulation community studies carefully live-migration performances to provide accurate cloud simulators. The migration models of [20], [25] assume an average memory dirty page rate that is refined during the simulation by the analysis of the prediction errors. Our approach predict the migration duration statically by a preliminary analysis of the VMs load. We modeled the memory dirty page generation in a two-stage process based on the analysis of common workloads observation. Haikun *et al.* [19] propose a good migration performance model based on the memory dirty page transfer algorithm implemented in Xen. They consider both static and refined dirty page rate build on historical observations and assume that the *Writable Working Set* size should be transferred in one round thereby determining the VM downtime. In contrast, we modeled the dirty page rate using a two-stage approach based on KVM behavior and we consider a preset maximum downtime for each VM migration. However, contrary to mVM they do not tackle migration scheduling and network topology that are the main contributions of this paper.

The CloudSim simulator [14] provides a migration model to estimate the migration duration but the model relies on the assumptions of Beloglasov *et al.* [6] discussed previously. Takahiro *et al.* [26] implement the pre-copy migration algorithm in the Simgrid simulator. They reproduces the memory dirty page generation behavior but they induced it from the CPU utilization with a proportional correlation between them. In contrast, we defined the dirty page generation rate statically, as a two-stage process, according to live VM memory observations and independently of the CPU usage. Sherif *et al.* [20] propose a simulator to reproduce the Xen migration algorithm with two different models. The first one is based on a constant average memory dirty page rate. The second model is a dynamic algorithm that learn from previous observations.

The aforementioned algorithms predict live-migration durations under different assumptions. To the best of our understanding, our model embraces the particularities of these algorithms without their possible restrictions. None of these models are however devoted to be used to compute quality migration schedules. [20], [25] reduce prediction errors with a feedback loop. They might have a better accuracy than our model, however, such an approach is not compatible with the need to compute a migration plan. Furthermore, our experimentations over a real network tested already reports 88.8% of the migrations scheduled by mVM have a 5 seconds slowdown at maximum against their theoretical minimal duration.

Scheduling live-migrations: [27], [10] study the factors that must be considered to schedule live-migrations efficiently. While Ye *et al.*[27] focused on resource reservation techniques on the source and the destination servers, we focused on the network topology and the dirty page rate [10]. These two works discuss about different scheduling policies that should be considered for the development of a migration scheduler. However, none of them proposed that scheduler. mVM is the migration scheduler that results from [10].

To the best of our knowledge, only Sarker *et al.* [25]

already propose an *adhoc* heuristic to schedule migrations. The objective is to reduce the completion time according to the network topology and a fixed dirty page rate. The heuristic is only compared to a custom algorithm that schedules the migrations randomly with regards to their theoretical completion time. The accuracy of the migration model and its practical benefits are not validated on a real testbed. We propose with mVM a migration model based on a two-stage process deduced from the practical observations of the workload, our scheduler can be enhanced to support additional constraints and we evaluated its prediction and benefits on a real testbed.

VII. Conclusion

Live-migration are used on a daily basis by consolidation algorithms and datacenter operators to manage the VMs on production servers. Current VM managers compute a placement of quality but usually neglect the main factors that impact the migration duration. This leads to unnecessarily long and costly migrations, prevents any control, and consumes an excessive amount of energy. We proposed mVM, a migration scheduler that infers the best moment to start the actions and the amount of bandwidth to allocate to them with regards to the VM workload, the network topology and user-specific constraints. mVM is implemented as a set of extensions for the VM manager BtrPlace in place of the old scheduler.

The accuracy of the migration model and the resulting decision capabilities of mVM have been validated through experiments on a real network testbed compared to the original scheduler of BtrPlace [13] and a scheduler that mimics Memory Buddies [15] decisions. Micro-experiments have shown that mVM outperforms both schedulers. On 50 migration plans generated randomly, migrations scheduled by mVM completed on average 20.4% faster than Memory Buddies, while completion times are reduced by 28.1%. Contrarily to Memory Buddies, mVM always outperforms sequential scheduling with a completion time speedup of 54.18% in average, while migration durations are close to the optimal with an average duration slowdown of 7.35% only, 4.5 times lower than with Memory Buddies. Thus making mVM more reliable.

Macro-experiments also exhibited the practical interest of mVM to address energy concerns. On a server decommissioning scenario involving 96 migrations among 72 servers having their ToR switches connected by a 10 Gbit/s aggregation switch, mVM reduced the energy consumption of the operation by 21.5% compared to BtrPlace. We also validated the control capacity of mVM by capping the power consumption of a schedule. Depending on the budget, mVM delayed migrations or server state switches to guarantee the power consumption remains below the given threshold. Finally, a scalability simulation reported mVM only requires 1.5 additional seconds with regards to BtrPlace to compute a schedule for the decommissioning scenario scaled by a factor of 10.

As a future work we want to consider the downtime as a variable of the model to infer when it is preferable to perform a cold migration over a live migration depending on the environment condition and the VM SLA. We finally want to merge the scheduler with the placement model of BtrPlace. Indeed, some schedules might be considered sub-optimal with respect to placement algorithm expectations in terms of reactivity. For example, the scheduler can introduce a delay to a migration as a consequence of a bad choice in terms of destination server. With a tight coupling between the two models, the placement algorithm will be able to revise its placement with respect to the scheduling decisions.

Availability

mVM is available as a part of the BtrPlace scheduler under the terms of the LGPL license. It can be downloaded, along with all the material related to the reproduction of the experiments at http://www.btrplace.org.

Acknowledgments

This work has been carried out within the European Projects DC4Cities (FP7-ICT-2013.6.2). Experiments presented in this paper were carried out using the Grid'5000 experimental testbed [23], being developed by INRIA with support from CNRS, RENATER and several universities as well as other funding bodies.

References

[1] "Service Level Agreement," http://cloud.google.com/compute/sla, 2015.

[2] "EC2 SLA," http://aws.amazon.com/fr/ec2/sla/, 2013.

[3] C. Clark, K. Fraser, S. Hand, and *al.*, "Live migration of virtual machines," in *Proceedings of the 2nd NSDI*. USENIX Assoc., 2005.

[4] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a Consolidation Manager for Clusters," in *VEE*. NY, USA: ACM, 2009.

[5] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *Middleware '08*. Springer-Verlag NY, Inc., 2008, pp. 243–264.

[6] A. Beloglazov and R. Buyya, "Energy Efficient Resource Management in Virtualized Cloud Data Centers," in *Proc. of the 0th IEEE/ACM Intl. Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 826–831.

[7] A. Verma, J. Bagrodia, and V. Jaiswal, "Virtual Machine Consolidation in the Wild," in *Middleware'14*. New York, USA: ACM, 2014.

[8] "Maintenance behavior," https://cloud.google.com/compute/docs/instances, 2015.

[9] Dean, Jeff, "Designs, Lessons and Advice from Building Large Distributed Systems," in *Keynote of the International Conference on Large-Scale Distributed Systems and Middleware Conference*, 2009.

[10] V. Kherbache, E. Madelaine, and F. Hermenier, "Planning Live-Migrations to Prepare Servers for Maintenance," in *Euro-Par: Parallel Processing Workshops*. Springer, 2014.

[11] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, and Z. Liu, "COMMA: Coordinating the Migration of Multi-tier Applications," in *VEE*. NY, USA: ACM, 2014.

[12] X. Wang and Y. Wang, "Coordinating power control and performance management for virtualized server clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 2, pp. 245–259, 2011.

[13] F. Hermenier, J. Lawall, and G. Muller, "BtrPlace: A Flexible Consolidation Manager for Highly Available Applications," *IEEE Trans. on Dependable and Secure Computing*, 2013.

[14] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. 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*, vol. 41, no. 1, 2011.

[15] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory Buddies: Exploiting Page Sharing for Smart Colocation in Virtualized Data Centers," in *Proc. of the ACM Intl. Conference on Virtual Execution Environments*, NY, USA, 2009.

[16] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: the Linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[17] N. Jussien, G. Rochart, and X. Lorca, "Choco: an Open Source Java Constraint Programming Library," in *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, Paris, France, 2008, pp. 1–10.

[18] A. Aggoun and N. Beldiceanu, "Extending CHIP in order to solve complex scheduling and placement problems," *Mathematical and Computer Modelling*, vol. 17, no. 7, pp. 57–73, 1993.

[19] H. Liu, H. Jin, C.-Z. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines." *Cluster Computing*, vol. 16, no. 2, 2013.

[20] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the Performance of Virtual Machine Migration." in *MASCOTS*, 2010.

[21] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy Live Migration of Virtual Machines," *SIGOPS OSR*, vol. 43, no. 3, pp. 14–26, Jul. 2009.

[22] J. Carlier, "The one-machine sequencing problem," *European Journal of Operational Research*, vol. 11, no. 1, pp. 42–47, 1982.

[23] R. Bolze, F. Cappello, M. Caron, Daydé, and *al.*, "Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed," *Int. Journal of High Performance Computing Applications*.

[24] VMware Inc, "vSphere Documentation Center," https://pubs.vmware.com/vsphere-51/topic/com.vmware.vsphere.vcenterhost.doc/GUID-3B41119A-1276-404B-8BFB-A32409052449.html, September 2015.

[25] T. Sarker and M. Tang, "Performance-driven live migration of multiple virtual machines in datacenters," in *IEEE International Conference on Granular Computing*, 2013.

[26] T. Hirofuchi, A. Lèbre, and L. Pouilloux, "Adding a Live Migration Model into SimGrid: One More Step Toward the Simulation of Infrastructure-as-a-Service Concerns," in *IEEE 5th Intl. Conference on Cloud Computing Technology and Science*, vol. 1, 2013, pp. 96–103.

[27] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, "Live migration of multiple virtual machines with resource reservation in cloud computing environments," in *IEEE International Conference on Cloud Computing*. IEEE, 2011, pp. 267–274.