Open access • Journal Article • DOI:10.1109/TCC.2017.2754279

Scheduling Live Migration of Virtual Machines — Source link

Vincent Kherbache, Eric Madelaine, Fabien Hermenier

Institutions: French Institute for Research in Computer Science and Automation, University of Nice Sophia Antipolis

Published on: 01 Jan 2020 - IEEE Transactions on Cloud Computing (IEEE)

Topics: Live migration and Schedule

Scheduling Live Migration of Virtual Machines

Vincent Kherbache, Eric Madelaine, Fabien Hermenier

Scheduling Live Migration of Virtual Machines

Vincent Kherbache, *Member, IEEE,* Éric Madelaine, *Member, IEEE,* and Fabien Hermenier, *Member, IEEE*

Abstract—Every day, numerous VMs are migrated inside a datacenter to balance the load, save energy or prepare production servers for maintenance. Although VM placement problems are carefully studied, the underlying migration schedulers rely on vague *adhoc* models. This leads to unnecessarily long and energy-intensive migrations.
We present mVM, a new and extensible migration scheduler. To provide schedules with minimal completion times, mVM parallelizes and sequentializes the migrations with regards to the memory workload and the network topology. mVM is implemented as a plugin of BtrPlace and its current library allows administrators to address temporal and energy concerns. Experiments on a real testbed shows mVM outperforms state-of-the-art migration schedulers. Compared to schedulers that cap the migration parallelism, mVM reduces the individual migration duration by 20.4% on average and the schedule completion time by 28.1%. In a maintenance operation involving 96 VMs migrated between 72 servers, mVM saves 21.5% Joules against BtrPlace. Compared to the migration model inside the cloud simulator CloudSim, the prediction error of the migrations duration is about 5 times lower with mVM. By computing schedules involving thousands of migrations performed over various fat-tree network topologies, we observed that the mVM solving time accounts for about 1% of the schedule execution time.

Index Terms—Live Migration, Scheduling, Virtual Machines

1 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 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. 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 [9]. 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 (i.e. time to complete all the migrations). On the opposite, an excessive parallelism leads to a low per-migration bandwidth allocation hence long or even endless migrations. Additionally, 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 [10], while a datacenter must also cap its power usage to fit the availability of renewable energies or ensure power cooling capabilities [11]. 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 pages 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 [12].

The evaluation of mVM is performed over a blocking network testbed against two representative schedulers: An unmodified BtrPlace that maximizes the migration parallelism similarly to [4], [6], [13], and a scheduler that reproduces Memory Buddies [14] decisions by statically capping the parallelism. The migration model accuracy is finally evaluated against representative cloud simulators models such as CloudSim [13] that tend to simplify the actual migration behavior, and SimGrid [15] which provides more realistic results. Our main results are:

Prediction accuracy: On 50 migration plans generated randomly, mVM estimated the migration durations with a precision of 93.9%. This is a 26.2% improvement over those computed by the model inside CloudSim [6], [13], BtrPlace [12] or Entropy [4] and an 1.7% improvement over SimGrid [16].

Migration speed: On the same random migration plans, the migrations scheduled by mVM completed on average 20.4% faster than Memory Buddies, while completion times were 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: The computation time of mVM to schedule thousands of migrations over various fat-tree network topologies accounts for less than 1% of the completion time.

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 2 describes the design of mVM. Section 3 details its implementation and Section 4 presents performance optimizations. Section 5 evaluates mVM. Finally, Section 6 describes related work, and Section 7 presents our conclusions and future works.

2 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 [17]. 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. We finally describe the integration of our mVM scheduler into BtrPlace.

2.1 Global design

Figure 1 depicts the architecture of mVM. mVM takes as input three types of information, 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]

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

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

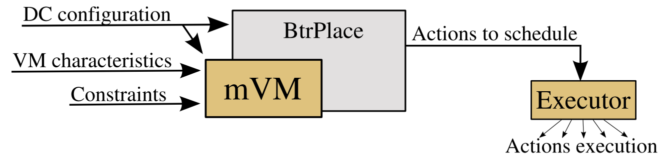


Fig. 1. mVM architecture.

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 a server that is not yet online. To address this issue, the executor inserts dependencies between actions according to a global virtual clock.

2.2 mVM integration

Figure 2 illustrates the interaction between BtrPlace and mVM. mVM only focuses on migration scheduling. Thus, It needs to know the destination server of each migration. We then integrated the mVM scheduling model by using a two rounds resolution. First, the original BtrPlace is used to compute a viable placement for each VM. Then, mVM retrieves the destination chosen for each migration and rely both on the network model to compute the bandwidth to allocate for each migration and on the migration model to estimate the corresponding migration duration.

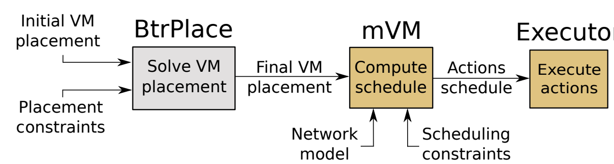


Fig. 2. Integration of mVM within BtrPlace using a two-round resolution.

3 IMPLEMENTATION

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

3.1 BtrPlace architecture

BtrPlace [12] 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 [18] 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 does not depend on 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 to each variable 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 moments 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.

Inside the core RP, mVM inserts 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.

3.2 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 migrates 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 $n \in \mathscr{N}$, $capa(n,t)$ denotes its capacity in Mbit/s at time t. For any migration $m \in \mathscr{M}$, $path(m) \subseteq \mathscr{N}$ indicates the network elements crossed (source and destination servers included), $bw(m,t)$ denotes the allocated bandwidth in Mbit/s at time t, $st(m)$ and $ed(m)$ indicate the beginning and the end of the operation in seconds, respectively. The equation (1) models the bandwidth sharing of a network element among the migrations that pass through it:

$$\sum_{\substack{m \in \mathscr{M},\ n \in path(m), \\ t \in [st(m);\ ed(m)]}} bw(m,t) \quad \leq \quad capa(n,t) \tag{1}$$

The bandwidth sharing is modeled with cumulative constraints [19]. Each consists in placing a set of tasks on a bounded resource. A task aggregates 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 defined two different elements that compose the network model. The first one is the *network link*. To represent both half- and full-duplex links, we use one or two cumulative constraints to represents its capacity depending on its duplex communication system. Indeed, when using half-duplex link, the communication is one direction at a time and it thus only requires one cumulative constraint. The full-duplex link, because of its two-way communication channel, requires two separate cumulative constraints to represent each direction that can be used simultaneously by different migrations. The second network element is the *switch*, which is used to connect the links together. In the case of a blocking-switch only, a single cumulative constraint is necessary to represent its limited capacity.

3.3 Migration model

In this section, we first give some background related to the live-migration protocol. Then we analyse the memory consumption induced by a migration on server side. Finally, we discuss the VM memory activity impact on the migration and describe our migration model accordingly.

3.3.1 Live-migration algorithm

The migration model mimics the pre-copy algorithm [3] used in Xen and KVM and 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 and its state 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 3).

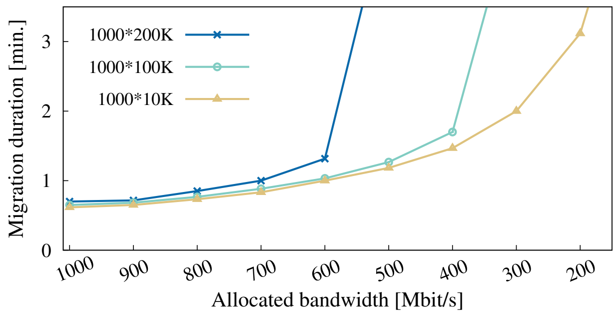


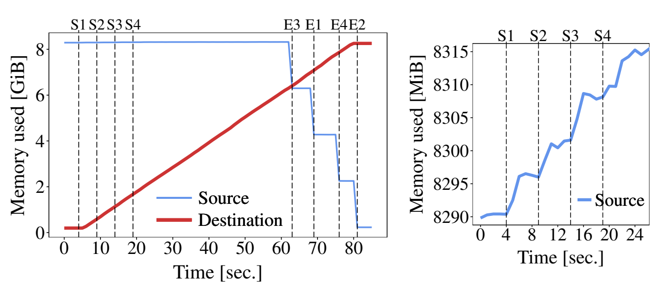
Fig. 3. Duration of a live-migration between 2 KVM hypervisors depending on the allocated bandwidth and the parameters used by the command *stress* to generate dirty pages. *1000*10K* indicates the VM runs 1000 concurrent threads that continuously allocate and release 10 KiB of memory each. The VM memory used is set to 4 GiB and the downtime is limited to 30 ms.

Using the *pre-copy* algorithm, the migration duration highly depends on the memory activity of the VMs to migrate. Additionally, as the memory used by a VM is mapped to the physical host memory, the behavior of servers' memory usage during a migration may impact its duration.

3.3.2 Server memory usage

To analyze the memory consumed by the migration and the memory transfer behavior between the servers, we migrated 4 VMs simultaneously from one server to another and monitored their memory usage. To actually observe the memory consumed by each migration separately, we started them by 5 seconds interval. The testbed consists of two identical servers with 16 GiB of memory and 2 quad-cores CPUs each. Every VM to migrate has a single vCPU and consumes 2 GiB of memory. To avoid extra migration duration, each VM has low CPU and memory activity (idle VMs). Figure 4 depicts the memory usage on the servers.

Figure 4a shows the memory consumption on the source and the destination servers during the experiment. The migrations are represented by the vertical dotted lines where *S1* to *S4* represent the start times and *E1* to *E4* the termination of the four migrations. As we can see, the source server releases the whole memory used by a VM at once just after the migration completed. This is



(a) Memory consumption overview (b) Memory used to migrate

Fig. 4. Overview of the memory consumption on both source and destination servers during 4 parallel migrations. Figure (b) is a closest view of the source server consumption when the migrations begin.

explained by the behavior of the pre-copy algorithm implemented on KVM. On the other side, the destination server receives the VM memory pages at the network speed of 1 Gbps and immediately fills them in memory. Its own memory consumption thus increases linearly during the migration. Consequently, this behavior does not require any post-migration delay due to guest memory mapping and is therefore not considered in the mVM migration model.

The memory usage due to the migration itself is too low to be observed on Figure 4a, we thus represented it more closely in Figure 4b. We observe that the memory consumption overhead induced by each migration is about 6 MiB which is negligible with regards to the memory available in current servers. Although this overhead may saturate the server, a common practice consists in reserving a sufficient amount of memory on the host to avoid such a saturation. Therefore, the additional memory used by the migration management is low enough to be safely ignored in the migration model.

3.3.3 VM memory activity

According to the majority of the loads observed on real applications [20], [21], the evolution of the memory dirtying rate can be separated in two phases. The first phase corresponds to the writing of the *hot-pages*, a set of memory pages that are quickly dirtied. This phase exhibits a high dirty pages rate but for a short period of time. The second phase represents the linear writing 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 give 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_s over a period equal to the downtime period D and ensuring that $\frac{HP_s}{D}$ is less than the available bandwidth on the migration path. The dirtying rate of the cold-pages CP_r in MiB/s can be measured after $t = HP_d$. Often very low, this rate is still dependent of the VM's workload. As the hot-pages, the cold-pages are rewritten at the beginning of the migration process. Thereby, the hot-pages dirtying rate is written: $HP_r = \frac{HP_s}{HP_d} - CP_r$. 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: $d_{min}(m) = \frac{mu(m)}{bw(m)}$. Hence if we assume that the total amount of cold-pages rewritten during the migration process CP_s is always lower than $mu(m)$, then CP_s can be written: $CP_s = d_{min}(m) \times CP_r$. In general, to transfer an amount of memory X with a bandwidth Y and a memory dirty rate Z, the transfer duration can be modeled by: $\frac{X}{Y-Z}$. Thus the time spent to send the cold-pages d_{CP} is written:

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

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

$$d_{HP}(m) = \frac{HP_s}{bw(m)} + \frac{HP_s - \big(D \times bw(m)\big)}{bw(m) - HP_r} \tag{3}$$

Where $\frac{HP_s}{bw(m)}$ corresponds to the first transmission of the hot-pages and $D \times bw(m)$ to the amount of data to send after suspending the VM on the source node (cf. during the downtime). If this value is higher than the measured amount of hot-pages ($D \times bw(m) > HP_s$), then it will not be necessary to iteratively send the hot-pages to comply with the desired downtime. In this case, the computation is thus simplified:

$$d_{CP}(m) + d_{HP}(m) = \frac{CP_s - \Big(\big(D \times bw(m) \big) - HP_s \Big)}{bw(m) - CP_r} + \frac{HP_s}{bw(m)} \quad (4)$$

Finally, the duration d of a migration m is:

$$d(m) = d_{min}(m) + d_{CP}(m) + d_{HP}(m) + D \quad (5)$$

The duration of d_{min} is the dominating factor. It is usually expressed in the order 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.

3.4 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.

3.4.1 Supporting the post-copy migration algorithm

Production oriented hypervisors implement the pre-copy migration algorithm. The post-copy algorithm is another approach that despite its efficiency, is not available in any production-oriented hypervisors [22]. It is worth noting that the extensibility of BtrPlace allows to support this migration model in place of the model discussed in Section 3.3.3. The post-copy algorithm consists in migrating the VM state and resume the VM at the beginning of the operation. The memory pages are then sent on-demand through the network. This tends to reduce the migration duration by removing the need to re-send dirty memory pages. Using mVM variables, this algorithm would be implemented by simply stating:

$$d(m) = \frac{mu(m)}{bw(m)} \quad (6)$$

3.4.2 Temporal control

Sync synchronizes the migrations of the VMs passed as parameters. It is a constraint inspired by COMMA [10]. 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 communication through a slow link. It is possible to migrate a VM using either a pre-copy or a post-copy algorithm. 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 pre-copy and post-copy approaches by synchronizing 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, 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* [23] 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:

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

3.4.3 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 [11]. 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.* [20]. 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 (7) 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.

$$\forall m \in \mathscr{M}, E(m) = \alpha \times bw(m) \times d(m) + \beta \quad (7)$$

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 [11], 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 ($a \in \mathscr{A}$), server ($s \in \mathscr{S}$) and VM ($v \in \mathscr{V}$) at every second of the reconfiguration. Indeed, the energy E consumed by an action corresponds to the sum of the instantaneous power P consumed at every second of its duration:

$$\forall a \in \mathscr{A}, \quad E(a) = \sum_{t \in [st(a), ed(a)]} P(a,t) \tag{8}$$

The objective is then implemented as follows:

$$\min \left(\sum_{a \in \mathscr{A}} E(a) + \sum_{s \in \mathscr{S}} E(s) + \sum_{v \in \mathscr{V}} E(v) \right) \tag{9}$$

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

4 Optimizing mVM

Computing the time to start each migration while satisfying temporal constraints and without exceeding the network elements capacities refers to the well-known *Resource-Constrained Project Scheduling Problem* (RCPSP), where each migration is an activity and each network element is a resource to share. However in our problem, each migration duration depends on its allocated bandwidth. The VM scheduling problem thus refers to a variant of the RCPSP called the *Multi-mode Resource-Constrained Project Scheduling Problem* (MRCPSP) where each activity has a set of available modes, and each mode corresponds to a couple of allocated bandwidth and the associated duration. Minimizing the makespan on both single and multi-mode RCPSP are known to be NP-hard [24], while MRCPSP is NP-Hard in the strong sense [25]. Therefore, computing a solution is time consuming when the number of VMs or the number of network elements 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 3.2, 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. By precomputing the bandwidth-duration couple for each migration, we thus reduced the problem complexity to a single-mode RCPSP.

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. The heuristic initially implemented for [26] appeared to be over specialised and only effective when addressing decommissioning scenarios. This prevented mVM to solve scheduling problems when the migrations where less ordered, subjects to dependencies or when servers must send and receive VMs. The new heuristic establishes three ordered groups of start moment variables: the servers boot actions, the migrations, and the nodes shutdown actions. Then the heuristic asks the solver to instantiate the start moments group by group. For the two groups of servers actions, the heuristic asks to focus on the hardest actions to schedule, *i.e.* those having the smallest domain for their start variable. For the group of migrations actions, the heuristic considers the migrations as a graph where servers are the vertices and the migrations are the arcs. It first forces the solver to focus on the migrations where the destination server is only subject to ongoing migrations. Then, it selects the migrations where the destination server has the lowest amount of outgoing migrations. This process is repeated until all the migrations start moments are ordered. Each time a start moment is selected by the search heuristic, the solver is forced to try its smallest possible value to start the actions 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 any instantiation that contradicts a constraint and the backtracking mechanism to revise a initial instantiation that turned to be invalid later in the search tree.

5 Evaluation

mVM aims at improving the live-migration scheduling thanks to an accurate migration model and appropriate reasoning. In this section, we first evaluate the practical benefits of mVM in terms of migration and reconfiguration speedup over a network testbed. We then evaluate the accuracy of the mVM migration model against different cloud simulators. Finally, we validate the capability of mVM to address energy concerns and evaluate its scalability and robustness by computing random migrations plans. Furthermore, all the experiments presented in this Section are reproducible [2].

On real experiments, we use the original BtrPlace and a scheduler derived from Memory Buddies [14] as representative baselines. We selected BtrPlace, first because its migration model is the same as other representative solutions [4], [6], [13] so their decision capabilities should be similar. Secondly, BtrPlace allows a precise comparison as the only software component that differs between mVM and BtrPlace is the scheduler, the core contribution of this paper. Memory Buddies scheduler provides also a relevant baseline as it allows to control the migration parallelism. Its approach consists in capping the amount of migrations to perform in parallel with a constant to be defined from the knowledge of the network topology.

On simulation experiments, we compare the migration estimation accuracy of mVM against the predictions made by the cloud simulator SimGrid [15] and two other representative migration models implemented in common cloud simulators such as CloudSim [13].

5.1 Testbed setup

All the real experiments were conducted on the Grid'5000 platform [27]. 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

2. https://github.com/btrplace

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 [28]. 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*.

5.2 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. To ease the reproducibility of our experiments, the memory workload for each VM is generated using the tool *stess* by running 1,000 threads that continuously write 70 KiB of RAM. This configuration allows to mimic the workload memory pattern of common applications types [20], [21]. We selected these specific values to reflect our measurements of the memory activity generated by two different HTTP benchmarks tools (*httperf* and *ab*). The memory activity details (including HP_s, HP_d, and CP_r) are defined in mVM from real measurements realised through our custom KVM patch developed for the occasion.

In this experiment, we compare the schedules computed by mVM against a scheduler that reproduces Memory Buddies [14] decisions. Similarly to mVM, Memory Buddies controls the migration parallelism, however it limits the parallelism to a constant to be defined. Memory Buddies migrates the VMs at least two at a time to fully exploit the two gigabit links when there are migrations between the servers connected by Gigabit interfaces and those connected by their emulated 500 Mbit/s interface. In practice, it is configured with three different parallelism setups that consist to migrate the VMs two, three, and four at a time. This parallelism setup is a good tradeoff that allows to exploit the full-duplex links capacity while limiting the risk of links saturation. 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. This prevent experiments from any bias due to a particular setup. Each run has been executed 3 times for each VM scheduler. In this experiment the original BtrPlace only computes full-parallel migrations plans due to its simplistic migration model. As this behavior prevents some migrations to terminate, we discarded the original BtrPlace scheduler for this experiment.

Table 1 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 times higher for MB-2. Figure 5 depicts the migration slowdowns as a *Cumulative Distribution Function* (CDF). We observe 88.8% of the migrations scheduled by mVM have a slowdown of 5 seconds at maximum, against 52.8% for MB-2. We also observe that the slowdown distribution for mVM is gathered while it is scattered for Memory Buddies and increasing with the concurrency.

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 revealed 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

TABLE 1
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%

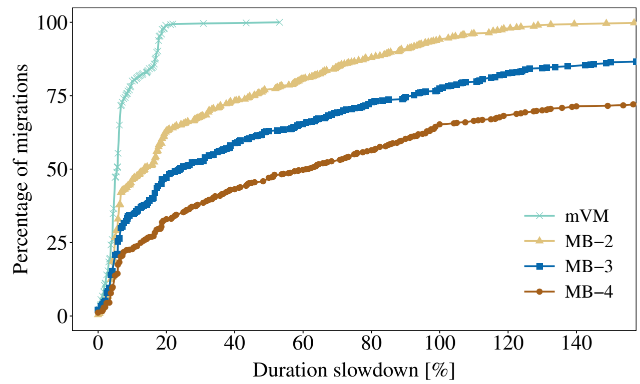


Fig. 5. CDF of migrations duration slowdown compared to sequential predictions. Migrations with a slowdown greater than 150% are not displayed.

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, but are not displayed due to a lack of space.

TABLE 2
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%

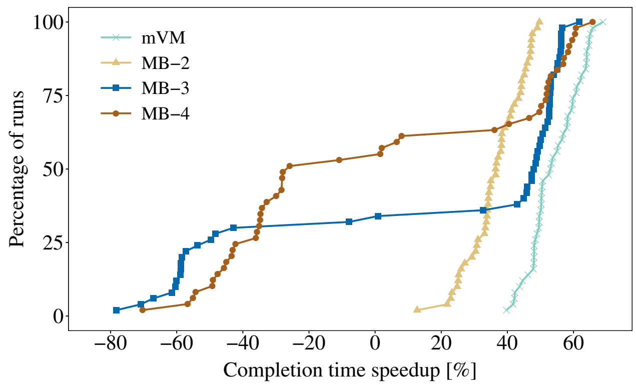


Fig. 6. CDF of completion times speedup compared to sequential executions.

Table 2 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 that the speedups for MB-3 and MB-4 are scattered and not always positives. This confirms that a blind over-parallelization of the migrations can produce longer completion times than a pure sequential scheduling. Indeed, as a consequence of the live-migration iterative behaviour, when the bandwidth on a network path is shared between too many migrations, the low bandwidth allocated to each migration leads to an exponential increase of their durations (see Section 3.3.1).

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 pages rate. Despite such an approach is a common practice in the state of the art and has already been tested under production workloads [9], [20], [21], some VMs might still have a fuzzy dirty pages rate. In this case the estimated migration duration might be inaccurate and 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.

5.3 Migration model accuracy

To assess the accuracy of our migration model, we compared the prediction deviation of common state-of-the-art cloud simulators against the real execution performed by mVM. Based on the same experiment setup than Section 5.2, we reproduced the scheduling decisions computed by mVM in all the selected simulators.

The first chosen migration model is representative of most common cloud simulators that ignore both the core network topology and the VMs workload. For instance, the well-used CloudSim [13] simulator, but also Entropy and BtrPlace do not considers the whole topology to migrate VMs but the servers network interfaces only. The VMs memory dirty pages rate is also ignored. We refer to this family of models as *NoShare*.

For the second migration model, namely *NoDP*, we decided to compute the migration time by only ignoring the workload running on VMs. The whole network topology is however considered, therefore this simulation model is only devoted to exhibit the migration deviation induced by ignoring the memory dirty pages rate of the VMs.

The third chosen migration model is the one implemented in the simulator SimGrid. It relies on a very realistic flow-based network model by using SFQ queuing mechanism, and considers full-duplex links and cross-traffic effects that impact on migration speed. The migration model also considers the VM memory dirty pages generation behavior [16]. It is represented by a dirty pages intensity modeled as a percentage of the available bandwidth tempered by the VM CPU usage. By considering a single constant rate of dirty pages, namely DP_r, the migration duration $d(m)$ is modeled by SimGrid using the following formula:

$$d(m) = \frac{mu(m)}{bw(m) - DP_r} \tag{10}$$

The dirty pages rate intensity DP_i is then obtained from the Equation (11) where $cu(m)$ is the CPU usage of the VM to migrate, in percentage:

$$DP_i = \frac{DP_r}{(cu(m) \times bw(m))} \tag{11}$$

The migration duration can thus be written:

$$d(m) = \frac{mu(m)}{bw(m) - (DP_i \times cu(m) \times bw(m))} \tag{12}$$

To setup SimGrid with representative parameters from our testbed, the CPU usage $cu(m)$ is set to 100% which corresponds to our *stress* workload. To retrieve a dirty pages intensity DP_i that also fits our workload, we first computed the amount of dirty

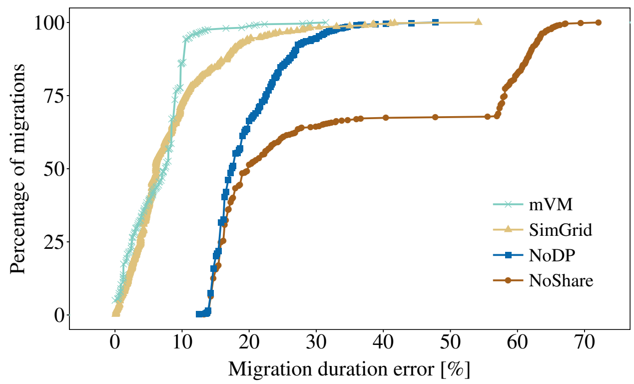


Fig. 7. CDF of normalized migration duration estimation error.

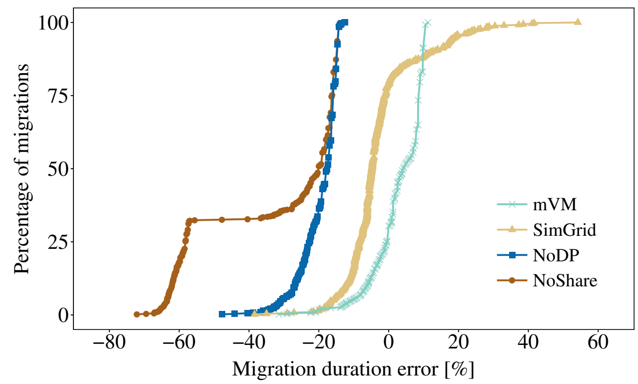


Fig. 8. CDF of relative migration duration estimation error.

pages generated during both the shortest and the longest observed migrations (resp. 20 to 61 seconds). We then deduced the two constant dirty pages rates that produce the equivalent amount of pages and retrieved the corresponding dirty pages intensity DP_i from the equation 11. We obtained a dirty pages intensity varying from 5% to 7% and selected the average of 6% as the global dirty pages intensity for each VMs.

The CDFs in Figure 7 show the normalized deviation of individual migration duration for each simulation environment. Precise values for all simulators are provided in Table 3.

TABLE 3
Normalised migration deviation compared to real execution

Migration model	NoShare	NoDP	SimGrid	mVM
Mean (%)	32.28	19.54	8.31	6.47
Std. dev. (%)	20.17	5.34	7.2	4.41
1st Quartile (%)	16.16	15.79	3.82	2.55
Median (%)	20	17.42	6.12	7.49
3d Quartile (%)	58.17	22.58	10.55	9.09

As expected, we observe that *NoShare* produces the longest migration deviations to reach 32.28% in average. The corresponding CDF is divided in two parts. 68% of the migrations are affected by only disregarding the VMs workload and the 32% others are mis-estimated due to both the workload and network sharing issues. In the last and worst case, the deviation ranges from 57% to 72% which obviously leads to strong practical scheduling issues.

NoDP produces long migration deviation with an average of 19.54%, three times higher than with mVM. Furthermore, its best prediction still induces a deviation of 10% against the real migration performed. This exhibits the importance of considering the memory activity of the VMs to estimate the migration duration.

By comparing SimGrid and mVM, we observe that the deviation are similar for 75% of the migrations. However, the average deviation is slightly slower using mVM where 95% of the migrations have at most 11% deviation against 77% for SimGrid. This demonstrates that the migration model of mVM, based on two distinct dirty pages rates (namely HP_r and CP_r), is more accurate than the single constant rate DP_r considered in SimGrid.

To analyze the migration deviation at a finer grain, the CDFs in Figure 8 show the distribution between under and over estimations. We first notice mVM provides stable results, the estimation deviation is balanced between low under- and over-estimation and never exceeds 11%.

Regarding SimGrid, we observe that the overall trend is to slightly underestimate the migration duration. This reflects its migration model, based on a single dirty pages rate, that tends to misestimate the actual overhead induced by the VMs memory activity. Indeed, by degrading our current model to a single rate, we undeniably lose accuracy by approximating a common value that will best suit all the migrations.

Additionally, despite a robust networking model, SimGrid tends to highly overestimate 20% of the migration durations to reach a 55% deviation in the worst case. This happens with migrations scenarios involving full-duplex network links usage in both uplink and downlink. We estimate that this overhead is due to an exaggeration of the cross traffic effects on the transfer speed.

The 5% negative outliers on all simulations are due to the traffic shaping setup in our testbed. Indeed in some particular cases, especially when the 500 Mbit/s links are used in full duplex, the migrations speed may decrease to 30% of its theoretical transfer speed induced by the traffic shaping. These outliers can be therefore ignored for the strict comparison of migration models accuracy.

5.4 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. Contrarily to BtrPlace, Memory Buddies cannot schedule the actions that consists in 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 by a 10 GBit/s aggregation switch. The decommissioning scenario consists in migrating the VMs from two racks to the third one. To save power, the destination servers are initially turned off and the servers 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. As BtrPlace tends to over-parallelize the migrations, all the VMs are set idle to prevent endless migration.

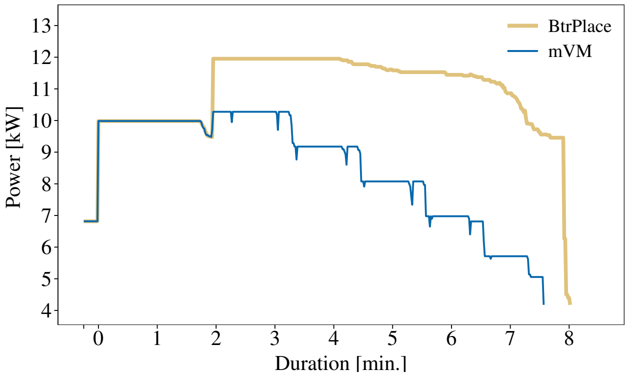


Fig. 9. Power consumption of migration plans.

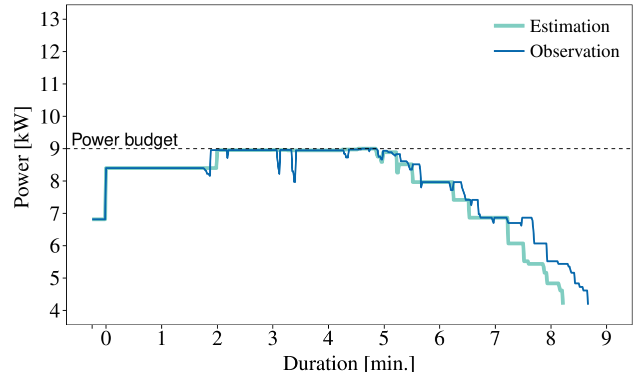


Fig. 10. Impact of a power capping on the power usage.

To calibrate the energy models with realistic values, we reused the experimental values from [20] for the migration energy model while the idle energy consumption of the servers were measured directly from the testbed (see Table 4).

TABLE 4
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

5.4.1 Energy saving capabilities

Figure 9 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 1.128 megajoules 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 9.

We observe mVM schedules the migrations 10 by 10. These groups were defined to maximize the bandwidth usage and minimize the migration duration. As stated in Section 4, the *MaxBandwidth* optimization forces a 1 Gbit/s bandwidth per migration, so the 10 by 10 parallelization fully uses 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 schedule and avoid overlapping actions. At the end, the completion time exceeded the prediction by only 5 seconds.

5.4.2 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 10 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 that 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 9).

Despite we measured a prediction accuracy of 93% for the migration durations, we observe that the practical completion time exceeds the prediction 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.

5.5 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. We successively evaluate the solving duration speedup of the *MaxBandwidth* optimization and the computational overhead of mVM against the original BtrPlace.

5.5.1 Experiment setup

To evaluate the scalability of the mVM scheduler, we used the same experiment setup as described in Section 5.4, and we scaled it up to 18 times using two different scaling factors.

The first scaling factor increases the infrastructure size. It is applied on the aggregation switch capacity and the number of racks to increase the number of network elements. At the largest scale (*x10*), each instance consists in scheduling the migration of 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. Regarding the mVM internals, this experiment evaluates the consequences of adding cumulative constraints and migration tasks.

The second scaling factor increases the amount of VMs in the infrastructure and the node hosting capabilities (memory and CPU resources). At the largest scale (*x18*), each instance requires to migrate 1728 VMs hosted on 2 racks of 24 servers each to a single rack. The consolidation ratio reaches 72 VMs per destination server and is also considered as exceptional for current datacenters. As an example, if each VM requires 4 virtual cores, this placement can even saturate the latest generation server of Bull, the bullion S, equipped with 288 cores on 16 Intel Xeon processor E7 v3. Regarding the mVM internals, this experiment evaluates the consequences of adding migration tasks.

To provide representative computation times, we generated 100 random instances for each scale and resolved each of them 10 times using both mVM and BtrPlace.

5.5.2 MaxBandwidth optimization

The *MaxBandwidth* optimization consists in only retaining the maximal bandwidth available on each migration path and its associated duration. Contextually, this optimization brings down the problem to a single-mode RCPSP which results in a significant reduction of the variables domain size and thus reduces the overall solving duration of mVM.

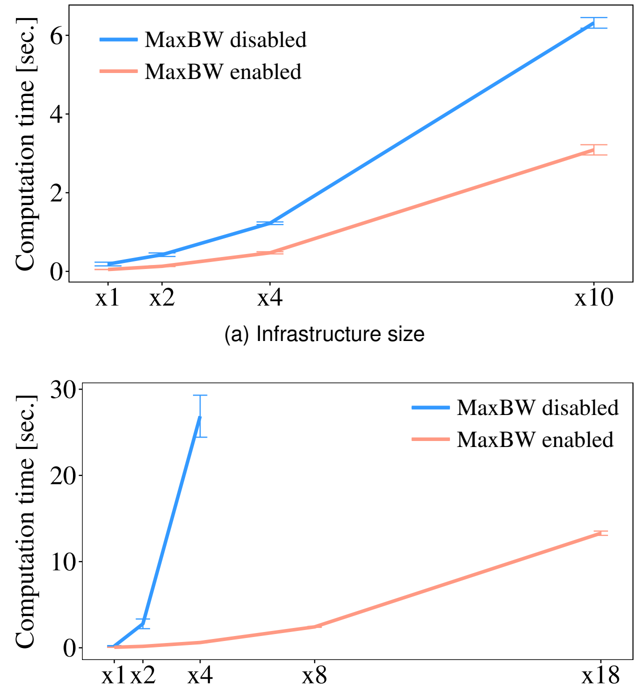
For these experiments, each node hosts up to 4 VMs and is connected to the network with a 1 Gbit/s link capacity. Therefore, the maximal amount of ongoing or outgoing migrations per node is 4. Hence, to allow a maximal parallelization of the migrations, we configured mVM to allow 4 different bandwidth allocations per migration, with steps of 250 Mbit/s when the *MaxBandwidth* optimization is disabled.

The random instances are generated by computing both random initial and final VMs placements among the predefined groups of source and destination nodes. Thus, despite the random placements, the scenario remains the same as the decommissioning experiment performed in Section 5.4.

We compare the computation time of mVM with the optimization enabled or disabled. Figure 11 shows the average computation time along with the 95% confidence intervals for each migration model and scaling factor. To ease data representation, we discarded the instances that required a computation time longer than a minute. The percentages of solved instances are however available in Table 5.

The results show that enabling the *MaxBandwidth* optimization reduces the computation time by up to 74% at the initial scale and increases the number of solved instances. When the infrastructure size increases (Figure 11a), the computation time for the two models increases exponentially but without deviating significantly. At the largest scale, enabling the optimization reduces the computation time by 3.2 seconds, a 51% speedup. Where the number of VMs increases (Figure 11b), only a few instances from scale *x1* to *x4* are solved in a minute when the option is disabled. This explains the larger confidence interval and the exponential increase of the solving duration. Enabling the optimization allowed to solve every instance in a minute with 43 times lower durations in average at scale *x4*.

Table 6b shows the amount of solved instances does not necessarily decrease when the size of the infrastructure increase. Indeed, at scale *x2* and with the optimization disabled, mVM solved only 22% of the instances and 83% at scale *x4*. This variability is explained by the random placements that lead to



(a) Infrastructure size

(b) Amount of VMs

Fig. 11. Comparison of the mVM solving duration depending on the *MaxBandwidth* optimization and the scaling factor. When the *MaxBandwidth* optimization is disabled, each migration has 4 different bandwidth allocations allowed by steps of 250 Mbit/s.

TABLE 5
Instances solved in a minute depending on the *MaxBandwidth* optimization and the scaling factor

Scale	*MaxBandwidth* option disabled	*MaxBandwidth* option enabled
x1	65%	100%
x2	22%	100%
x4	83%	100%
x10	62%	100%

(a) Infrastructure size

Scale	*MaxBandwidth* option disabled	*MaxBandwidth* option enabled
x1	66%	100%
x2	57%	100%
x4	47%	100%
x8	0%	100%
x18	0%	100%

(b) Amount of VMs

numerous cumulative constraints. In such conditions, a variable height and duration for the tasks may easily lead the solver to take wrong decisions on variable instantiations thereby drastically increasing the computation time.

To conclude, this experiment shows that forcing the bandwidth to allocate to the migrations and pre-computing their duration improve the performances with no counterparts.

5.5.3 mVM against Btrplace

In this experiment, we compare the time that is required to compute the schedule using mVM or BtrPlace with instances generated randomly. A preliminary scalability experiment was conducted in [26], however the solved instances only represented symmetric server decommissioning scenarios. We observed the initial heuristic was over specific and prevented mVM to solve randomly generated instances in a minute. The optimizations discussed in Section 4 fixed that robustness issue. Now every randomly generated instances are solved by mVM. Furthermore, to provide a more robust analysis of our migration model, in this experiment we computed random VMs placements among the entire set of nodes and thus without relying on a decommissioning scenario. Figure 12 shows the results.

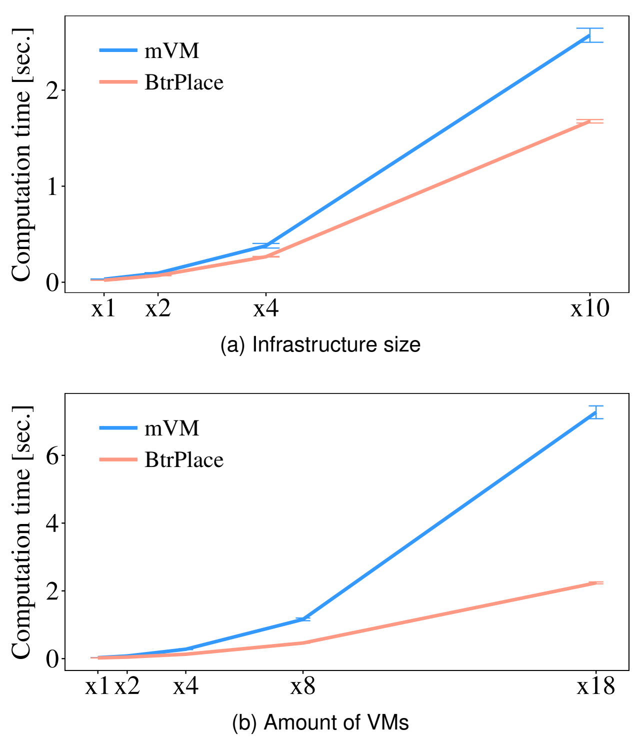


Fig. 12. Computation time of mVM and BtrPlace depending on the scaling factor

At the lowest scale, mVM takes 9.4 ms more than BtrPlace to compute a solution, that is a 48% increase. When the size of the infrastructure increases, the overhead increases up to 82%. At this scale, mVM requires 2.6 seconds to schedule 960 migrations between 720 servers. When the number of VMs increases, the overhead increases up to 231.8%. mVM requires then 7.3 seconds to schedule 1728 migrations between 72 servers. This overhead is explained by the additional computations made by mVM to provide reliable, fast, and energy efficient schedules in practice. Even if the relative overhead is significant, the time required to compute a schedule stays negligible with regards to the time that is required to execute it. Indeed, at the highest scales, the schedules computed by mVM are predicted to take from 152 to 224 sec. by increasing the number of VMs, and from 22 to 26 minutes by increasing the infrastructure size. Therefore, in the worst case the computation phase only represents 1.7% and 0.6% of the execution phase when the number of VMs and the infrastructure size are respectively scaled up.

We finally observe that at equivalent scaling factors, the overhead of mVM is bigger when the number of VMs increases. For example, at scale *x4* the computation time of mVM is 36.5% higher by scaling up the number of VMs rather than the infrastructure. This difference is explained by the network model. A cumulative constraint has a $\mathcal{O}(n^2)$ time-complexity where n is the number of tasks to schedule. This indicates that adding more tasks on the same cumulative constraints, *i.e.* adding more VMs per network element, is more computationally intensive than adding more cumulative constraints with the same amount of tasks.

We also globally observe slower computation times for mVM than with the decommissioning scenario in Figure 11. This is mainly explained by the network links usage. Indeed, while the number of migrations remains exactly the same, dissolving the static groups of source and destination nodes leads to a global lower consolidation. Therefore, the nodes links are used less intensively and the amount of tasks per cumulative constraints is reduced consequently. Additionally, unlike in the decommissioning scenario, each network link may also be used in full-duplex. This results in more cumulative constraints (*i.e.* one per link direction) with fewer migration tasks placed on them thereby reducing the overall computation time.

At a very large scale, the solving duration for mVM might become significant with regards to the completion time. A solution to overcome this limitation would be to split the operation in multiple steps. At the moment the bandwidth used to migrate VMs exceeds the 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.

6 Related works

6.1 Migration scheduling in VM managers

Many works such as [4], [6], [12], [13] estimate the migration duration to be equal to the VM memory usage divided by the network bandwidth. The experiments discussed in Section 5 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 [14] addresses the impact of concurrent live-migrations by capping the concurrency with a number to be defined. The experiments discussed in Section 5 also proved that this assumption is not optimal. Indeed, the concurrency cannot be constant as it depends on the current network load and the migration path. COMMA [10] considers the network bandwidth and the dirty pages 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 but with the knowledge of the whole topology.

[29] and [9] study the factors that must be considered to schedule live-migrations efficiently. While Ye *et al.* [29] focus on resource reservation techniques on the source and the destination servers, [9] focus on the network topology and the dirty pages rate. These two works discuss about different scheduling policies that should be considered for the development of a migration scheduler. However, none of them proposes that scheduler.

Sarker *et al.* [30] propose an *adhoc* heuristic to schedule migrations. The heuristic reduces the completion time according to the network topology and a fixed dirty pages 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 is 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.

Bari *et al.* [31] restrict parallelism to migrations with disjoint paths only. With regards to the topology of our experiments in Section 5.2, this lead to migrate up to 2 VMs in parallel while mVM migrate up to 5 VMs in parallel at full speed with a negligible overhead. Yao *et al.* [32] restrict parallelism within the same rack while guaranteeing at least 70 percent of maximum migration speed. Such a simplification would lead to purely sequential migrations in our experiments. Wang *et al.* [33] consider a non-blocking network and the multi-path routing feature made available by SDN controllers to increase the migration bandwidth. mVM supports blocking networks and does not rely on any SDN features. Overall, these algorithms are only evaluated through simulations with no guarantee over the simulator accuracy while we provide extensive experimentations on a real network testbed.

6.2 Predicting live-migration duration for simulation

The simulation community studies carefully live-migration performances to provide accurate cloud simulators. The migration models of [21], [30] assume an average memory dirty pages rate that is refined during the simulation by the analysis of the prediction errors. Our approach predicts the migration duration statically by a preliminary analysis of the VMs load. We model the memory dirty pages generation in a two-stage process based on the analysis of common workloads observation. Haikun *et al.* [20] propose a migration performance model based on the memory dirty pages transfer algorithm implemented in Xen. They consider both static and refined dirty pages 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 model the dirty pages rate using a two-stage approach based on KVM behavior and we consider a preset maximum downtime for each VM migration. They also do not tackle migration scheduling and network topology that are the main contributions of this paper.

The CloudSim simulator [13] provides a model to estimate the migration duration but the model relies on the assumptions of Beloglasov *et al.* [6] discussed previously. Takahiro *et al.* [16] implemented the pre-copy migration algorithm in the Simgrid simulator. They reproduce the memory dirty pages generation behavior by using a single rate but with unusual linear correlation on the CPU usage. In contrast, we define the dirty pages generation rate as a two-stage process, according to live VM memory observations and independently of the CPU usage. Sherif *et al.* [21] proposes a simulator to reproduce the Xen migration algorithm with two different models. The first one is based on a constant average memory dirty pages rate. The second model is a dynamic algorithm that learns 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 but not their limitations. None of these models are however devoted to be used to compute migration schedules. [21], [30] reduce prediction errors with a feedback loop. Such an approach is not compatible with the need to compute a migration plan.

7 Conclusion

Live-migrations 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, and consumes an excessive amount of energy. We propose 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 has been validated through random migrations plans simulation against the execution on a real testbed. We compared mVM predictions to the cloud simulator SimGrid [15] and two representative migrations models such as the one implemented in CloudSim [13] and the original BtrPlace [12]. Results show that mVM migration model is most accurate than any other with an average accuracy of 93.9%.

The scheduling decisions of mVM have been validated through experiments on a real network testbed compared to the original scheduler of BtrPlace and a scheduler that mimics Memory Buddies [14] decisions. Micro-experiments have shown that mVM outperforms both schedulers. On migration plans generated randomly, migrations scheduled by mVM completed 20.4% faster than Memory Buddies, with completion times reduced by 28.1%. Contrarily to Memory Buddies, mVM always outperforms sequential scheduling with a completion time speedup of 54.18%. Migration durations are close to the optimal with a slowdown of 7.35% only, 4.5 times lower than with Memory Buddies.

Macro-experiments validated the use 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 and server state switches to guarantee the power consumption remains below the given threshold. Finally, a scalability evaluation has shown that mVM is suitable to schedule thousands of migrations. By varying the consolidation ratio or the infrastructure size, we observed the computation time of mVM amounts for less than 1% of the completion time.

As a future work we 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. 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.

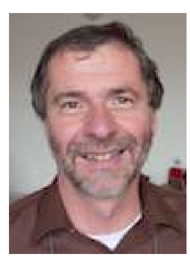
Acknowledgments

This work has been carried out within the European Projects DC4Cities (FP7-ICT-2013.6.2). Experiments were carried out using the Grid'5000 experimental testbed [27], 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*. ACM, 2009.
[5] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *Middleware*. 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*. IEEE, 2010.
[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] V. Kherbache, E. Madelaine, and F. Hermenier, "Planning Live-Migrations to Prepare Servers for Maintenance," in *Euro-Par: Parallel Processing Workshops*. Springer, 2014.
[10] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, and Z. Liu, "COMMA: Coordinating the Migration of Multi-tier Applications," in *VEE*. ACM, 2014.
[11] 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.
[12] F. Hermenier, J. Lawall, and G. Muller, "BtrPlace: A Flexible Consolidation Manager for Highly Available Applications," *IEEE Trans. on Dependable and Secure Computing*, 2013.
[13] 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.
[14] 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 *VEE*, 2009.
[15] 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*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014.
[16] 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.
[17] 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.
[18] 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*, 2008.
[19] 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.
[20] H. Liu, H. Jin, C.-Z. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines." *Cluster Computing*, 2013.
[21] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the Performance of Virtual Machine Migration." in *MASCOTS*, 2010.
[22] 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.
[23] J. Carlier, "The one-machine sequencing problem," *European Journal of Operational Research*, vol. 11, no. 1, pp. 42–47, 1982.
[24] J. Blazewicz, J. Lenstra, and A. Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11 – 24, 1983.
[25] R. Kolisch, A. Sprecher, and A. Drexl, "Characterization and generation of a general class of resource-constrained project scheduling problems," *Management science*, vol. 41, no. 10, pp. 1693–1703, 1995.
[26] V. Kherbache, E. Madelaine, and F. Hermenier, "Scheduling live-migrations for fast, adaptable and energy-efficient relocation operations." in *IEEE/ACM Intl. Conference on Utility and Cloud Computing*, 2015.
[27] 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*.
[28] VMware Inc, "vSphere Documentation Center," Sep 2015.
[29] 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.
[30] T. Sarker and M. Tang, "Performance-driven live migration of multiple virtual machines in datacenters," in *IEEE International Conference on Granular Computing*, 2013.
[31] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "CQNCR: Optimal VM migration planning in cloud data centers," in *Networking Conference, 2014 IFIP*. IEEE, 2014, pp. 1–9.
[32] X. Yao, H. Wang, C. Gao, F. Zhu, and L. Zhai, "VM migration planning in software-defined data center networks," in *IEEE International Conference on HPCC, SmartCity, and DSS*, Dec 2016, pp. 765–772.
[33] H. Wang, Y. Li, Y. Zhang, and D. Jin, "Virtual machine migration planning in software-defined networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 487–495.

Vincent Kherbache received his Ph.D in Computer Science from the University of Côte d'Azur in 2016 (France). His research interests focus on virtualized resources management, adaptive maintenance operations and energy-efficiency in renewable-powered Data Centres. Since 2017, he is working as an R&D Engineer to provide fault tolerance and high resiliency over multiple IaaS when executing heterogeneous workflows. To this end, he is mainly interested in concurrent programming and container orchestration.



Éric Madelaine is a senior researcher at INRIA in Sophia-Antipolis, France. He has an engineer diploma from Ecole Polytechnique, Paris, a PhD in computer science from Univ. of Paris 7, and an Habilitation from Univ. of Nice. His research interests range from semantics of programming languages, distributed and cloud computing, component-based software, formal methods, methods and tools for specification and verification of complex programs.

Fabien Hermenier received the Ph.D degree in 2009 from the University of Nantes. He has been an associate professor at University of Nice Sophia-Antipolis since 2011. His research interests are focused on cloud computing and resource management. Since 2006, he has been working on virtual machine placement algorithms in IaaS infrastructures.