Higher SLA Satisfaction in Datacenters with Continuous VM Placement Constraints

Huynh Tu Dang
University Nice Sophia Antipolis, CNRS, I3S, UMR 7271, France
hdang@polytech.unice.fr

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

ABSTRACT

In a virtualized datacenter, the Service Level Agreement for an application restricts the Virtual Machines (VMs) placement. An algorithm is in charge of maintaining a placement compatible with the stated constraints.

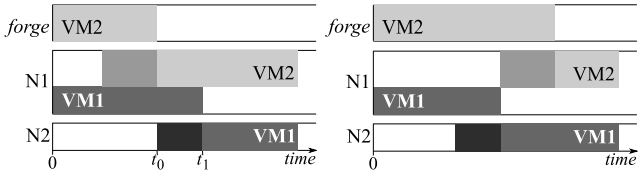
Conventionally, when a placement algorithm computes a schedule of actions to re-arrange the VMs, the constraints ignore the intermediate states of the datacenter to only restrict the resulting placement. This situation may lead to temporary violations of the constraints. In this paper, we discuss the causes of these violations. We then advocate for continuous placement constraints to restrict also the actions schedule. We discuss why their development requires more attention and how the extensible placement algorithm BtrPlace can address this issue.

1. INTRODUCTION

A datacenter provides a platform of choice for an economical hosting of applications. Thanks to virtualization and consolidation techniques, applications can be embedded in Virtual Machines (VMs) and deployed on demand on a reduced number of nodes [11]. In practice, a user requests several VMs and expresses the quality of service he expects using a Service Level Agreement (SLA). Such a SLA consists in placement constraints asking for a minimum amount of resources to ensure performance. It may also express placement expectations such as requesting distinct nodes for VMs running service replicas to achieve fault tolerance.

A VM placement algorithm is in charge of finding a node for every VM in the datacenter according to the stated constraints [2, 9, 10, 13, 14, 17, 18]. Virtualization enables a number of operations to improve continuously the datacenter infrastructure and the performance of the VMs. This includes dynamic allocation of resources to the VMs, node maintenance, failure recovery or VM packing [16]. In these contexts, the placement algorithm computes a *reconfiguration plan* describing the schedule of actions (migration[7], startup, shutdown, *etc.*) to execute to re-arrange the VMs.

A common behavior for a placement algorithms [3, 17, 13] consists is to select the new hosting node for every VM with respect to the placement constraints, then to schedule the actions heuristically as early as possible. Such a *discrete approach* does not exclude temporary violations that may occurred during the reconfiguration, as illustrated by the following example: Figure 1 illustrates this situation through two reconfiguration plans where a constraint denied the colocation of VM1 and VM2. Initially, VM1 is running on N1. VM2 is waiting for being deployed and can only run on N1 due to its resource requirement. Figure 1a depicts a reconfiguration plan satisfying a discrete anti-colocation only. The VMs are not colocated at the end of the reconfiguration but they overlap between t_0 and t_1. Figure 1b depicts a continuous anti-colocation: The arrival of VM2 is delayed until VM1 leaves N1 to disallow any overlap.



(a) discrete satisfaction (b) continuous satisfaction

Figure 1: Two reconfiguration plans leading to 2 non-colocated running VMs. VM2 is originally on a *forge*, waiting for being launched on a node.

A SLA may be defined in terms of uptime across periods. This is required to be compatible with unpredictable hardware failures, but also with variable performance isolation [19]. The temporary violation caused by a discrete constraint is not unpredictable. It is caused by a perfectible definition of the constraint that results from a lack of vision about the possible situations during a reconfiguration process. In this context, temporary violations are avoidable through a finer definition of the constraints. In this paper, we first demonstrate through the simulation of realistic reconfiguration scenarios, how discrete placement constraints may lead to temporary violations. We then propose *continuous constraints* to disallow these temporary violations by controlling the actions schedule, hence the VMs and the nodes at any moment of their lifetime. We provide a preliminary implementation of continuous constraints and val-

*Fabien Hermenier is a member of OASIS, a joint project with Inria, the CNRS I3S and the University of Nice Sophia Antipolis.

HotDep'13, November 03 - 06 2013 Farmington, PA, USA

idate their ability to improve the reliability of the datacenter. We finally observe that the development of continuous constraints requires more care with regards to their discrete equivalent.

The remainder of this paper is organized as follows. Section 2 describes related works. Section 3 evaluates the reliability of discrete placement constraints. Section 4 presents continuous placement constraints and the challenges we face to define and implement them. Section 5 presents our conclusions and future research directions.

2. RELATED WORK

Entropy [14] and Wood *et al.* [20] consider the actions schedule in their placement algorithms. They justify this need to solve dependencies between migrations. Their ad-hoc heuristics to treat the actions scheduling disallow however the addition of any placement constraints. Bin *et al.* [2] provide the k-resilient property for a VM in case of node failure. The algorithm provides a continuous high-availability by allowing to migrate only the VMs marked as highly-available. This approach disallows temporary violations but limits the management capabilities of the VMs to this particular use case.

Authors in [3, 13, 17] propose algorithms that support an extensible set of placement constraints. They however rely on a discrete approach to compute the placement. In [13, 17], a node is choosed for every VM randomly until one satisfies the placement constraints at the end of the reconfiguration. The actions required to achieve this new placement are then scheduled heuristically at their earliest time, with no consideration of the continuous satisfaction of the constraints. Breigand *et al.* [3] use a score function to choose the best node for each VM. The same temporary overlap may then appear if putting the first VM on the node hosting the second one leads to a better score.

Despite the library of BtrPlace [13] is composed by discrete constraints only, it provides to the developers the necessary material to manipulate the actions schedule. For this reason, we rely on it 1) to exhibit the temporary violations that are allowed by the use of discrete constraints and 2) to provide a preliminary implementation of continuous constraints.

3. EVALUATING THE RELIABILITY OF DISCRETE PLACEMENT CONSTRAINTS

We evaluate here the reliability of discrete placement constraints available in the VM placement algorithm BtrPlace [13]. We simulate a datacenter subject to common reconfiguration scenarios and inspect the computed reconfiguration plans to exhibit temporary violations.

3.1 Simulation Setup

The simulated datacenter consists of 256 nodes split over 2 clusters. Each cluster consists of 8 racks of 16 nodes each. Inside a rack, nodes are connected through a non-blocking network. Each node has 128 GB RAM and a computational capacity of 64 uCPU, a unit similar to ECU in Amazon EC2 [1] to abstract the hardware. We simulate 350 3-tier web applications, each having a structure generated randomly to use between 6 and 18 VMs, with 2 to 6 VMs per tier. In total, the datacenter runs 5,200 VMs. Table 1 summarizes the VM resource consumption for each tier. VMs in the Tier 1 require a few and balanced amount of resources. VMs in the Tier 2 and 3 are computation and memory intensive, respectively. By default, each VM consumes only 25% of the maximum allowed. This makes the datacenter uCPU and memory usage equal to 68% and 40% respectively.

Tier	Initial consumption		Max. consumption	
	uCPU	RAM	uCPU	RAM
1	1	1 GB	2	4 GB
2	4	2 GB	14	7 GB
3	1	4 GB	4	17 GB

Table 1: VM characteristics

A SLA is attached to each application. For each tier, one `spread` constraint requires distinct nodes for the VMs to provide fault tolerance. This mimics the VMWare DRS anti-affinity rule [10]. One `among` constraint forces VMs of the third tier to be hosted on a single rack to benefit from the fast interconnect. Last, 25% of the applications use a `splitAmong` constraint to separate the replicas among the two clusters. This mimics Amazon EC2 high-availability through location over multiple availability zones.

The datacenter is also subject to placement constraints. The maximum number of online nodes is restricted to 240 by a `maxOnline` constraint to simulate a hypervisor-based licensing model [6]. The 16 remaining nodes are offline and spread across the racks to prepare for application load spikes or hardware failures. Last, one `singleResourceCapacity` constraint preserves 4 uCPU and 8 GB RAM on each node for the hypervisor management operations.

On the simulated datacenter, we consider four reconfiguration scenarios that mimic industrial use cases described in[16]:

`Vertical Elasticity.` In this scenario, applications require more resources for their VMs to support an increasing workload. In practice, 10% of the applications require an amount of resources equals to the maximum allowed. With BtrPlace, this is expressed using `preserve` constraints to ask for at least a given amount of resources.

`Horizontal Elasticity.` In this scenario, applications require more VMs to support an increasing workload. In practice, 25% of the applications doubles the size of their tiers. With BtrPlace, this is expressed using `running` constraints to ask to run more VMs.

`Hardware Failure.` Google reports [8] that yearly, a network outage makes 5% of the nodes instantly unreachable. This scenario is simulated by setting randomly 5% of the nodes in a failure state and asking to restart their VMs on other nodes.

`Boot Storm.` In virtual desktop infrastructures, numerous desktop VMs are started simultaneously before the working hours. To simulate this scenario, 400 new VMs are required to boot. Their characteristics are chosen randomly among those described in Table 1 and their resource consumption is set to the maximum allowed.

Each scenario is simulated through 100 different instances that initially satisfy all the constraints. For ear instance, every scenario introduces a disruption that is required to be fixed by BtrPlace. The simulator runs on a dual CPU Xeon X5570 at 2.93 GHz with 24 GB RAM that runs Linux

Scenario	Violated SLAs	Actions			
		VM Boot	Migrate	Node Boot	Node Shutdown
Vertical Elasticity	40.72	0%	99.99%	0.005%	0.005%
Horizontal Elasticity	0.19	99.82%	0.18%	0%	0%
Server Failure	29.56	61.29%	35.89%	2.82%	0%
Boot Storm	0.35	98.57%	1.43%	0%	0%

Table 2: Violated SLAs and actions composing the reconfiguration plans.

2.6.32-5-amd64 and Sun's JVM 1.7.0_25.

Table 2 summarizes the average number of SLAs in each instance that are not fulfilled, *i.e.* where at least one constraint is violated during the reconfiguration, as well as the distribution of the actions that compose the reconfiguration plans. We observe that the `Vertical Elasticity` and the `Server Failure` scenarios lead to the most violations. Furthermore, we observe a relation between the number of migrations and the number of SLA violations. Initially, the current placement of the running VMs satisfies the constraints. During a reconfiguration, some VMs are migrated to their new hosts and once all the migrations are terminated, the resulting placement also satisfies the constraints. Until the reconfiguration is complete, a part of the VMs may be on their new hosts while the others are still waiting for being migrated. This leads to an unanticipated combination of placements that was not under the control of the constraints. This explanation is also valid for constraints that manage the node state, such as `maxOnline`. When the reconfiguration plan consists of booting VMs only, this situation cannot occur with the studied set of constraints. The placement constraints ignore non-running VMs and as each VM is deployed on a satisfying host, a temporary violation is not possible.

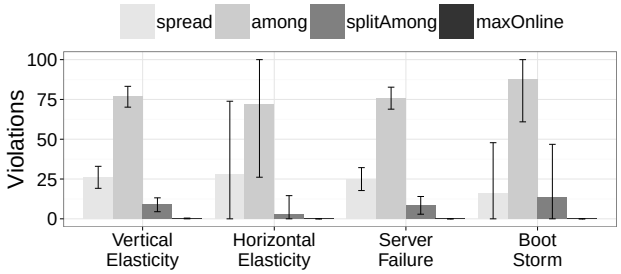


Figure 2: Distribution of the violated constraints

Figure 2 presents the average distribution of violated constraints. We observe `among` is the most violated constraint. Furthermore, with a distribution that exceeds 100%, we observe multiple constraints can be violated within a single SLA. The violations are explained by the behavior of BtrPlace. To compute a small reconfiguration plan, BtrPlace tries to keep each VM on its current node, otherwise, it chooses a new satisfying host randomly to balance the load. If a VM involved in an `among` constraint must be migrated, BtrPlace has 94% chances to select a node on another rack, hence violating the constraint until the other involved VMs are relocated to this rack. This violation may reduce the performance of the application due to the slow interconnect between the racks. For a VM subjects to a `spread` constraint, there is 2% chances at worst to select a node that already hosts a VM involved in the same constraint. The constraint will then be violated, leading to a potential single point of failure, until the other VM has been relocated elsewhere. For a VM subjects to a `splitAmong` constraint, the chances to select a node in the other cluster are 50% and the consequences of the resulting violation are similar to `spread`. Figure 2 reports more violations of `spread` than `splitAmong` constraints. There are 14 times more `spread` constraints so the chances to violate at least one constraint are higher.

Over the 800 studied instances, the `maxOnline` constraint has been violated 3 times, all in the `vertical elasticity` scenario. In these cases, the load increase saturated some racks. BtrPlace chose to boot the rack's spare node to absorb the load and to shutdown a node in a non-saturated rack in exchange to satisfy the constraint at the end of the reconfiguration. BtrPlace scheduled the actions as soon as possible. It then decided to boot the spare node before shutting down the other node which leaded to a temporary violation of the constraint. With floating hypervisor licenses, this would lead to a non-applicable reconfiguration plan. The `singleResourceCapacity` constraint was never violated: BtrPlace schedules the VM migrations before increasing their resource allocation, it is then not possible to have a resource consumption that exceeds the host capacity.

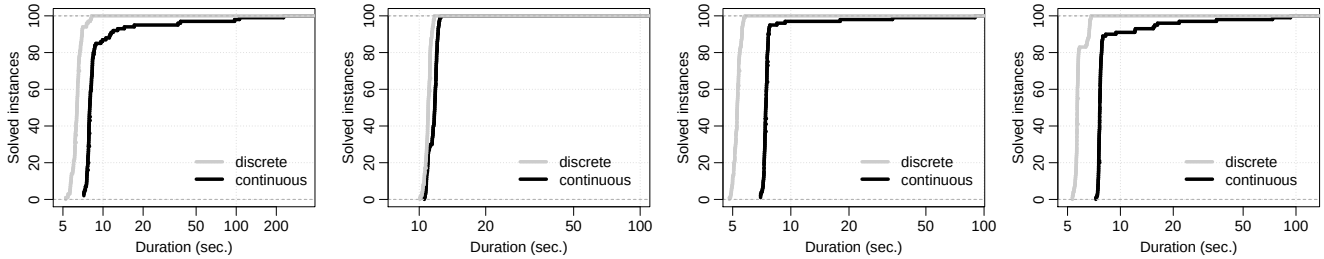
This experiment reveals temporary violations occur in BtrPlace despite the usage of placement constraints. We also discussed in Section 2 that other placement algorithms [3, 17] may also compute plans leading to temporary violations. While the consequences of these violations depend on the constraints, their cause is only related to the lack of controls over the computation of the actions schedule.

4. TOWARD CONTINUOUS RESTRICTION FOR PLACEMENT CONSTRAINTS

Depending on the users expectations, the permanent satisfaction of the placement constraints may be critical. However, we showed discrete placement constraints do not put restrictions over the temporal aspect of a reconfiguration. We thus propose *continuous constraints* to also control the actions schedule hence, the VMs and the nodes during their whole lifetime. We discuss here the implementation and evaluation of continuous version of the constraints evaluated in Section 3.

4.1 Implementing Continuous Constraints

The implementation of the continuous constraints relies on the extensibility capabilities provided by BtrPlace. BtrPlace provides a composable placement algorithm based on Constraint Programming (CP)[12]. In CP, a problem is modeled as a set of variables, each taking its value in a finite domain,



(a) Vertical Elasticity (b) Horizontal Elasticity (c) Server Failure (d) Boot Storm

Figure 3: Cumulated distribution of the solving durations per scenario

and a set of independent constraints that represents the required relations between the values of the variables. A solver computes a solution for a problem by assigning each variable to a value that simultaneously satisfies the constraints.

To implement a discrete constraint, a developer only pays attention to the future placement of the VMs and the next state of the nodes. To implement continuous constraints, he might also has to integrate the schedule of the associated actions. This requires a new look on the restrictions to express. What was a pure assignment problem becomes a scheduling problem that is harder to tackle.

BtrPlace already exposes variables related to the VM placement and node state at the end of the reconfiguration, but also variables related to the actions schedule. We have implemented the continuous constraints by extending the discrete ones to act, when necessary, on the variables related to the actions schedule.

The continuous implementation of `among` and `splitAmong` does not need to manipulate the actions schedule. If a VM is already running on a node, the continuous implementation just enforces the placement of the associated VMs the associated group.

The continuous implementation of `spread` prevents the VMs from overlapping. In practice, when a VM is migrated to a node hosting another VM involved in the same constraint, we force the start time of the migration action to be greater or equal to the variable denoting the moment the other VM will leave the node.

The discrete implementation of `maxOnline` extracts the boolean variables that indicate if the nodes must be online at the end of the reconfiguration and forces their sum to be at most equal to the given amount. For a continuous implementation, we model the online periods of the nodes and guarantee that their overlapping never exceeds the limit K.

BtrPlace does not provide variables to model the online period of a node. We have then defined this period from pre-existing variables related to the scheduling of the potential node action. We defined on_i and off_i as the moments a node i becomes online or offline, respectively. If the node is already online, on_i equals 0, *i.e.* the beginning of the reconfiguration process. Otherwise it is equal to the variable denoting the moment the node may start to go online. If the node is already offline, off_i is equal to the variable denoting the end of the reconfiguration process or the variable denoting the moment the node goes offline. The online period for the node i is then defined with a time interval bounded by on_i and off_i. We finally restrict the overlap of the online periods as follow:

$$\forall t \in \mathcal{T}, card(\{i | on_i \geq t \wedge off_i \leq t\}) \leq K \quad (1)$$

This restriction can be implemented as a *cumulative* constraint[4], available in Choco [5], the CP solver used in BtrPlace. *Cumulative* is a common scheduling constraint that restricts the overlapping of tasks that have to be executed on a resource. The continuous implementation of `maxOnline` appeared to be more challenging than the others. The developer must have a deeper understanding of the node actions to model their online period. For an efficient implementation, he must also understand how his problem can be expressed using pre-existing constraints.

4.2 Preliminary results

The additional restriction provided by the continuous constraints may impact the performance of the underlying placement algorithm. We evaluate here their computational overhead with regards to their discrete equivalent on the simulator presented in Section 3.

Figure 3 presents as a cumulated distribution function the solving duration for the instances in every scenario. As expected, none of the plans computed using continuous constraints leads to a temporary violation.

With only discrete constraints, BtrPlace computed a solution for each instance in 12 seconds maximum with an homogeneous solving duration for each scenario. The use of continuous constraints introduces a computational overhead that vary from 0.5 seconds to 6.5 seconds depending on the scenario. This was however expected as the additional restrictions to disallow temporary violations necessarily increase the combinatorial complexity of the reconfiguration algorithm.

We also observe the usage of continuous constraints impacts the distribution of the solving time, especially in the `Vertical Elasticity` and the `Boot Storm` scenarios. In the `Vertical Elasticity` scenario, 90% of the instances are solved in less than 12 seconds, *i.e.* with a 5 seconds overhead. It however requires up to 200 seconds to solve 97% of the instances and the remaining 3% was still not solved after 600 seconds. This reveals a few instances are very hard to solve within a reasonable duration. Comparing these hard instances with the others may exhibit performance bottlenecks and lead to faster implementation of continuous constraints. We are currently studying the unsolved instances but we think they cannot be solved without a relaxation of the continuous component of some constraints.

5. CONCLUSION AND FUTURE WORK

In a virtualized datacenter, users express their SLAs with placement constraints. A VM placement algorithm is then in charge of placing the VMs on the nodes according to the stated constraints. Usually, a constraint controls the VM placement only at the end of the reconfiguration process and ignores the datacenter intermediary states between the beginning and the end of the reconfiguration process.

In this paper, we advocated this discrete approach is not sufficient to satisfy the SLAs continuously as an uncontrolled actions schedule may indeed lead to temporary violations. We exhibited these violations through BtrPlace, one of the placement algorithms that relies on discrete placement constraints. We observed the violations occur when a reconfiguration plan aims at fixing a resource contention or fragmentation using migrations. We then proposed continuous constraints to control the quality of service during the reconfiguration by restricting the actions schedule. We implemented preliminary version of continuous constraints and confirmed they improve the datacenter reliability by removing any temporary violations. We also observed the implementation of these constraints is more challenging for the developers. It first requires detailed understanding of the principles of a reconfiguration. It also transforms a supposed placement problem into a scheduling one which may be harder to tackle. Finally, the computational complexity of scheduling problems may reduce significantly the scalability of the placement algorithm.

We want to keep investigating on the different situations that lead to temporary violations. Energy-efficient placement algorithms rely heavily on VM migrations. These algorithms are widely used and should be studied as they shall be subject to such violations. We also want to consider placement constraints that address new concerns to investigate for their likelihood of being subject to violations, and for the resulting consequences.

We observed that continuous constraints impact the performance of a placement algorithm, while not being constantly required to deny temporary violations. We want then to explore the situations that make continuous placement constraints unnecessary and rely on a static analysis of the problems to detect these situations.

Finally, continuous constraints may lead to unreasonable solving durations for hard instances or, even worse, unsolvable problems. In these situations, a user may prefer a temporary violation of his SLA. This however implies to allow the users to characterize their tolerance. We want then to model the cost of a constraint violation and integrate this notion inside BtrPlace.

Acknowledgments

This work was partially supported by the OpenCloudware project. OpenCloudware is funded by the French Fonds national pour la Société Numérique (FSN), and is supported by Pôles Minalogic, Systematic and SCS.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies.[1]

[1] `http://www.grid5000.fr`

Availability

BtrPlace is licensed under the terms of the LGPLv3 License[15]. All the material related to the reproduction of the experiments is available at `http://btrp.inria.fr`.

6. REFERENCES

[1] Amazon EC2. `http://aws.amazon.com/ec2/`.

[2] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz. Guaranteeing high availability goals for virtual machine placement. *2012 IEEE 32nd International Conference on Distributed Computing Systems*, 0:700–709, 2011.

[3] D. Breitgand, A. Marashini, and J. Tordsson. Policy-driven service placement optimization in federated clouds. *IBM Research Division, Tech. Rep*, 2011.

[4] Y. Caseau and F. Laburthe. Cumulative scheduling with task intervals. In *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 363–37, 1996.

[5] Choco: an open source Java constraint programming library. Research report 10-02-INFO, Mines de Nantes, 2010.

[6] Citrix store. `http://store.citrix.com`.

[7] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *2nd NSDI*, pages 273–286, 2005.

[8] 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.

[9] C. Dupont, T. Schulze, G. Giuliani, A. Somov, and F. Hermenier. An energy aware framework for virtual machine placement in cloud federated data centres. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, pages 4:1–4:10, New York, NY, USA, 2012. ACM.

[10] D. Epping and F. Denneman. *VMware vSphere 4.1 HA and DRS technical deepdive*. CreateSpace, 2010.

[11] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCS '03, pages 550–, Washington, DC, USA, 2003. IEEE Computer Society.

[12] F. Hermenier, S. Demassey, and X. Lorca. Bin repacking scheduling in virtualized datacenters. *Principles and Practice of Constraint Programming–CP 2011*, pages 27–41, 2011.

[13] F. Hermenier, J. Lawall, and G. Muller. Btrplace: A flexible consolidation manager for highly available applications. *IEEE Transactions on Dependable and Secure Computing*, 10(5):273–286, 2013.

[14] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 41–50, New York, NY, USA, 2009. ACM.

[15] Gnu lesser general public license version 3. `https://www.gnu.org/licenses/lgpl-3.0.en.html`.

[16] V. Soundararajan and J. M. Anderson. The impact of management operations on the virtualized datacenter. *SIGARCH Comput. Archit. News*, 38(3):326–337, June 2010.

[17] K. Tsakalozos, M. Roussopoulos, and A. Delis. Hint-based execution of workloads in clouds with nefeli. *Parallel and Distributed Systems, IEEE Transactions on*, 24(7):1331–1340, 2013.

[18] A. Verma, P. Ahuja, and A. Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 243–264, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[19] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 1163–1171, Piscataway, NJ, USA, 2010. IEEE Press.

[20] 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 *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 31–40, New York, NY, USA, 2009. ACM.