HAL
open science

Energy Management in IaaS Clouds: A Holistic Approach

Eugen Feller, Cyril Rohr, David Margery, Christine Morin

To cite this version:

Eugen Feller, Cyril Rohr, David Margery, Christine Morin. Energy Management in IaaS Clouds: A Holistic Approach. [Research Report] RR-7946, INRIA. 2012. hal-00692236

HAL Id: hal-00692236

https://inria.hal.science/hal-00692236v1

Submitted on 29 Apr 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Energy Management in IaaS Clouds: A Holistic Approach

Eugen Feller, Cyril Rohr, David Margery, Christine Morin

RESEARCH REPORT

N° 7946

April 2012

Project-Teams MYRIADS

ISSN 0249-6399 ISRN INRIA/RR--7946--FR+ENG

Energy Management in IaaS Clouds: A Holistic Approach

Eugen Feller*, Cyril Rohr *, David Margery *, Christine Morin *

Project-Teams MYRIADS

Research Report n° 7946 — April 2012 — 21 pages

Abstract: Energy efficiency has now become one of the major design constraints for current and future cloud data center operators. One way to conserve energy is to transition idle servers into a lower power-state (e.g. suspend). Therefore, virtual machine (VM) placement and dynamic VM scheduling algorithms are proposed to facilitate the creation of idle times. However, these algorithms are rarely integrated in a holistic approach and experimentally evaluated in a realistic environment.

In this paper we present the energy management algorithms and mechanisms of a novel holistic energy-aware VM management framework for private clouds called Snooze. We conduct an extensive evaluation of the energy and performance implications of our system on 34 power-metered machines of the Grid'5000 experimentation testbed under dynamic web workloads. The results show that the energy saving mechanisms allow Snooze to dynamically scale data center energy consumption proportionally to the load, thus achieving substantial energy savings with only limited impact on application performance.

Key-words: Cloud Computing, Energy Management, Consolidation, Relocation, Live Migration, Virtualization

* INRIA Centre Rennes - Bretagne Atlantique, Campus universitaire de Beaulieu, 35042 Rennes, France - {Eugen Feller, Cyril.Rohr, David.Margery, Christine Morin}@inria.fr

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Gestion d'énergie pour les services informatiques hébergés sur le mode IaaS: une approche intégrée

Résumé : La performance énergétique est maintenant devenue l'une des contraintes majeures pour les opérateurs actuels et futurs de centres de cloud. Une des manières de conserver l'énergie est de faire passer les serveurs inutilisés dans un état de consommation moindre (par exemple, 'suspend'). Par conséquent, des algorithmes de placement et d'ordonnancement dynamique de machine virtuelle (MV) ont été proposés pour faciliter la création de périodes d'inactivité. Cependant ces algorithmes sont rarement intégrés dans une solution complète, et rarement évalués de manière expérimentale dans un environnement réaliste.

Dans cet article, nous présentons les algorithmes et mécanismes de gestion d'énergie de Snooze, un système novateur de gestion de MV pour centres de cloud privés. Nous effectuons une évaluation approfondie des implications en terme d'énergie et de performance de ce système en reproduisant une charge typique des applications web dynamiques, sur 34 machines de la plateforme d'expérimentation Grid'50000, dont la consommation en énergie peut être mesurée. Les résultats montrent que les mécanismes de conservation d'énergie de Snooze lui permettent d'adapter la consommation énergétique d'un centre de cloud proportionnellement à la charge, conduisant ainsi à des gains significatifs en terme de consommation énergétique, avec un impact limité sur les performances de l'application.

Mots-clés : Informatique en nuage, Gestion d'énergie, Consolidation, Relocalisation, Migration à chaud, Virtualisation

1 Introduction

Cloud computing has gained a lot of attention during the last years and cloud providers have reacted by building increasing numbers of energy hungry data centers in order to satisfy the growing customers resource (e.g. storage, computing power) demands. Such data centers do not only impose scalability and autonomy (i.e. self-organization and healing) challenges on their management frameworks, but also raise questions regarding their energy efficiency [1]. For instance, Rackspace which is a well known Infrastructure-as-a-Service (IaaS) provider hosted approximately 78.717 servers and served 161.422 customers in 2011 [2]. Moreover, in 2010 data centers have consumed approximately 1.1 - 1.5% of the world energy [3].

One well known technique to conserve energy besides improving the hardware is to virtualize the data centers and transition idle physical servers into a lower power-state (e.g. suspend) during periods of low utilization. Transitioning idle resources into a lower power state is especially beneficial as servers are rarely fully utilized and *lack power-proportionality*. For example, according to our own measurements conducted on the Grid'5000 experimental testbed in France, modern servers still consume a huge amount of power ($\sim$ 182W) despite being idle. Consequently, taking energy saving actions during periods of low utilization appears to be attractive and thus is the target of our research. However, as virtual machines (VMs) are typically load balanced across the servers, idle times need to be created first. Therefore, dynamic VM relocation and consolidation can be used in order to migrate VMs away from underutilized servers. It can be done either event-based (i.e. relocation) upon underload detection or periodically (i.e. consolidation) by utilizing the live migration features of modern hypervisors (e.g. KVM [4], Xen [5]).

Some dynamic VM relocation (e.g. [6]) and many consolidation algorithms (e.g. [7, 8, 9]) have been recently proposed with only few of them being validated in a realistic environment (e.g. [6]) though under static workloads (i.e. the number of VMs in the system stays constant). Moreover, all these works either target relocation or consolidation and mostly consider only two resources (i.e. CPU, memory).

To the best of our knowledge none of the mentioned works: (1) integrate most of the energy management mechanisms within a holistic cloud management framework: VM resource utilization monitoring and estimations, overload and underload anomaly detection, relocation, consolidation, and power management; (2) experimentally evaluate them under dynamic workloads (i.e. on-demand VM provisioning); (3) consider more than two resource dimensions (e.g. CPU, memory, network Rx, and network Tx).

In our previous work [10] we have proposed a novel scalable and autonomic VM management framework for private clouds called Snooze. In this work, we focus on its energy management algorithms and mechanisms. Our first contribution is a unique holistic solution to perform VM resource utilization monitoring and estimations, detect and react to anomaly situations and finally do dynamic VM relocation and consolidation to power off and on idle servers. Our second contribution is an experimental evaluation of the proposed algorithms and mechanisms in a realistic environment using dynamic web workloads on 34 power-metered nodes of the Grid'5000 experimental testbed.

The results show that Snooze energy management mechanisms allow it to

scale the data center energy consumption proportionally to current utilization with only limited impact on application performance, thus achieving substantial energy savings. This work has direct practical application as it can be either applied in a production environment to conserve energy or as a research testbed for testing and experimenting with advanced energy-aware VM scheduling algorithms.

The remainder of this article is organized as follows. Section 2 discusses the related work. Section 3 introduces the energy saving algorithms and mechanisms of Snooze. Section 4 presents the evaluation results. Section 5 closes this article with conclusions and future work.

2 Background

Energy conservation has been the target of research during the last years and led to many works at all levels (i.e. hardware and software) of the infrastructure. This section focuses on the software level and presents related work on VM relocation and consolidation.

In [8] multiple energy-aware resource allocation heuristics are introduced. However, only simulation-based results based on simple migration and energy-cost models are presented. Finally, only one resource dimension (i.e. CPU) is considered.

In [11] the authors propose a multi-objective profit-oriented VM placement algorithm which takes into account performance (i.e. SLA violations), energy efficiency, and virtualization overheads. Similarly to [8] this work considers CPU only and its evaluation is based on simulations.

In [6] a framework is introduced which dynamically reconfigures a cluster based on its current utilization. The system detects overload situations and implements a greedy algorithm to resolve them. However, it does not include any energy saving mechanisms such as underload detection, VM consolidation and power management.

In [9] a consolidation manager based on constraint programming (CP) is presented. It is solely limited to static consolidation (i.e. no resource overcommitment is supported) and neither includes overload/underload anomaly detection, relocation, nor any power saving actions.

In [12] the Eucalyptus cloud management framework is extended with live migration and consolidation support. The extension neither supports anomaly detection nor event-based VM relocation. Moreover, it remains unclear when and how many migrations are triggered during its evaluation. Finally, it targets static workloads and is tested on three nodes which is far from any real cloud deployment scenario.

Last but not least in [13] the VMware Distributed Resource Scheduler (DRS) is presented. Similarly, to our system DRS performs dynamic VM placement by observing the current resource utilization. However, neither its system (i.e. architecture and algorithms) nor evaluation (i.e. performance and energy) details are publicly available.

Snoozes goes one step further than previous works by providing a unique *experimentally evaluated holistic energy management approach for IaaS clouds.*

3 Energy Management in IaaS Clouds: A Holistic Approach

Snooze is an energy-aware VM management framework for private clouds. Its core energy conservation algorithms and mechanisms are described in this section.

First we introduce the system model and its assumptions. Afterwards, a brief overview of the system architecture and its parameters is given. Finally, the energy management algorithms and mechanisms are presented.

3.1 System Model and Assumptions

We assume a homogeneous data center whose nodes are interconnected with a high-speed LAN connection such as Gigabit Ethernet or Infiniband. They are managed by a hypervisor such as KVM [4] or Xen [5] which supports VM live migration. Power management mechanisms (e.g. suspend, shutdown) are assumed to be enabled on the nodes. VMs are seen as black-boxes. We assume no restriction about applications: both compute and web applications are supported.

3.2 System Architecture

The architecture of the Snooze framework is shown in Figure 1 . It is partitioned into three layers: physical, hierarchical, and client.

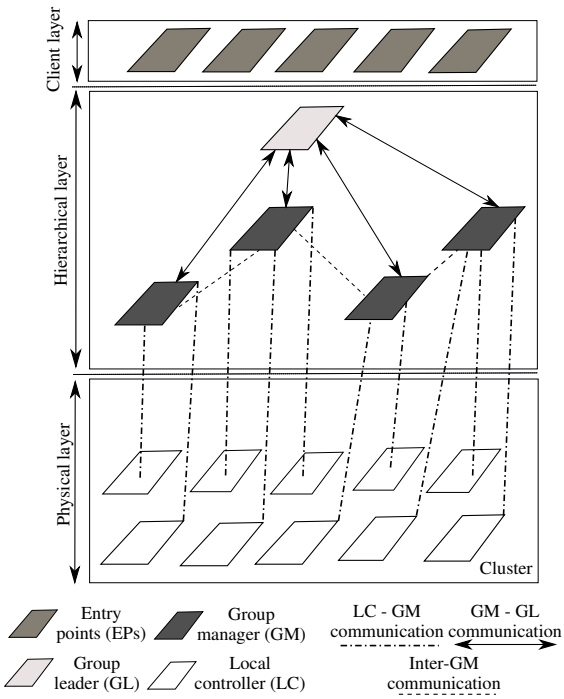

Figure 1: System Architecture

At the physical layer, machines are organized in a cluster, in which each node is controlled by a so-called *Local Controller (LC)*.

A hierarchical layer allows to scale the system and is composed of fault-tolerant components: *Group Managers (GMs)* and a *Group Leader (GL)*.

Each GM manages a *subset* of LCs and is in charge of the following tasks: (1) VM monitoring data reception from LCs; (2) Resource (i.e. CPU, memory and network) utilization estimation and VM scheduling; (3) Power management; (4) Sending resource management commands (e.g. start VM, migrate VM, suspend host) to the LCs.

LCs enforce VM and host management commands coming from the GM. Moreover, they monitor VMs, detect overload/underload anomaly situations and report them to the assigned GM.

There exists one GL which oversees the GMs, keeps aggregated GM resource summary information, assigns LCs to GMs, and dispatches VM submission requests to the GMs. The resource summary information holds the total active, passive, and used capacity of a GM. Active capacity represents the capacity of powered on LCs, while passive capacity captures resources available on LCs in power saving state. Finally, used capacity represents the aggregated LC utilization.

A client layer provides the user interface. It is implemented by a predefined number of replicated *Entry Points (EPs)*.

3.3 System Parameters

Let LCs denote the set of LCs and VMs the set of VMs, with n = $|LCs|$ and $m = |VMs|$ representing the amounts of LCs and VMs, respectively.

Available resources (i.e. CPU, memory, network Rx, and network Tx) are defined by the set R with d = $|R|$ (d = 4). CPU utilization is measured in *percentage of the total LC capacity*. For example, if a LC has four physical cores (PCORES) and a given VM requires two virtual cores (VCORES), the maximum CPU requirement of a VM would be 50%. Memory is measured in Kilobytes and network utilization in Bytes/sec.

VMs are represented by requested and used capacity vectors ($\mathbf{RC_v}$ resp. $\mathbf{UC_v}$). $\mathbf{RC_v} := \{RC_{v,k}\}_{1 \leq k \leq d}$ reflects the static VM resource requirements in which each component defines the requested capacity for resource k $\in$ R. They are used during the initial VM submission to place VMs on LCs. On the other hand, used capacity vectors $\mathbf{UC_v} := \{UC_{v,k}\}_{1 \leq k \leq d}$ become available as the result of monitoring. Each component of the vector represents the estimated VM utilization for resource k over the last measurement period T (e.g. one day).

LCs are assigned with a predefined static homogeneous capacity vector $\mathbf{C}_l := \{C_{l,k}\}_{1 \leq k \leq d}$. In addition, their current utilization $\mathbf{c}_l$ is computed by summing up the VM used capacity vectors: $\mathbf{c}_l := \sum_{\forall v \in LC_l} \mathbf{UC}_v$.

We introduce $\mathbf{M}_l := \{MID_{l,k}\}_{1 \leq k \leq d}$ as the LC resource capping vector which puts an *upper bound on the maximum aimed LC utilization for each resource k* with $0 \leq MID_{l,k} \leq 1$. In other words we keep a limited amount of available resources to compensate for overprovisioning. This is required in order to mitigate performance problems during periods of high resource contention.

LC_l is considered to have enough capacity for VM_v if either $\mathbf{c}_l + \mathbf{RC_v} \leq \mathbf{C}_l \diamond \mathbf{M}_l$ holds during submission or $\mathbf{c}_l + \mathbf{UC_v} \leq \mathbf{C}_l \diamond \mathbf{M}_l$ during VM relocation or consolidation. $\diamond$ denotes elementwise vector multiplication.

Introducing resource upper bounds leads to situations where VMs can not be hosted on LCs despite enough resources being available. For example when $MID_{l,CPU} = 0.8$ and only two PCORES exist, VM requiring all of them can not be placed (i.e. 2 VCORE / 2 PCORE ≤ 0.8 does not hold).

Therefore, we define the notion of packing density (PD) which is a vector of values between 0 and 1 for each resource k. It can be seen as the *trust given to the user's requested VM resource requirements* and allows VMs to be hosted on LCs despite existing MID capping's. When PD is enabled, Snooze computes the requested VM resource requirements as follows: $\mathbf{RC_v} := \mathbf{RC_v} \diamond \mathbf{PD}$.

In order to detect anomaly situations we define a $0 \leq MIN_k \leq 1$ and $0 \leq MAX_k \leq 1$ threshold for each resource k. If the estimated resource utilization for k falls below MIN_k the LC is considered as underloaded, otherwise if it goes above MAX_k LC it is flagged as overloaded (see the following paragraphs).

LCs and VMs need to be sorted by many scheduling algorithms. Sorting vectors requires them to be first normalized to scalar values. Different sort norms such as L1, Euclid or Max exist. In this work the L1 norm is used.

3.4 Resource Monitoring and Anomaly Detection

Monitoring is mandatory to take proper scheduling decisions and is performed at all layers of the system. At the physical layer VMs are monitored and resource utilization information is periodically transferred to the GM by each LC. It is used by the GM in the process of VM resource utilization estimation and scheduling.

At the hierarchical layer, each GM periodically sends aggregated resource summary information to the GL. This information includes the used and total capacity of the GM with the former being computed based on the estimated VM resource utilization of the LCs and is used to guide VM dispatching decisions.

Overload and underload anomaly detection is performed locally by each LC based on aggregated VM monitoring values. This allows the system to avoid many false-positive anomaly alerts. Particularly, for each VM a system administrator predefined amount of monitoring data entries is first collected. After the LC has received all VM monitoring data batches, it performs the total LC resource utilization estimation by averaging the VM resource utilizations and summing up the resulting values. Finally, a threshold crossing detection (TCD) is applied on each dimension of the estimated host resource utilization vector based on the defined MIN_k and MAX_k thresholds to detect anomaly situations. LCs are marked as overloaded (resp. underloaded) in the data sent to GM if at least one of the dimensions crosses the thresholds.

3.5 Resource Utilization Estimations

Resource utilization estimations are essential for most of the system components. For example, they are required in the context of anomaly detection and VM scheduling (i.e. placement, relocation, and consolidation).

GM performs LC resource utilization estimations in order to generate its aggregated resource summary information. TCD decisions are based on es-

timated VM resource utilizations. Finally, in the context of VM scheduling, VM resource utilizations are estimated in order to: (1) Compute the total LC resource utilization; (2) Sort LCs and VMs.

Snooze provides abstractions which allow to easily plug in different estimators for each resource. For example VM CPU utilization can be estimated by simply considering the average of the n most recent monitoring values. Alternatively, more advanced prediction algorithms (e.g. based on Autoregressive-Moving-Average (ARMA)) can be used. In this work the former approach is taken.

3.6 Energy-Aware VM Scheduling

Scheduling decisions are taken at two levels: GL and GM.

At the GL level, VM to GM dispatching is done based on the GM resource summary information. For example, VMs could be dispatched across the GMs in a capacity-aware round-robin or first-fit fashion. In this work round-robin is used. Thereby, GL favors GMs with enough active capacity and considers passive capacity only when not enough active one is available.

Note that summary information is not sufficient to take *exact dispatching decisions*. For instance, when a client submits a VM requesting 2GB of memory and a GM reports 4GB available it does not necessary mean that the VM can be finally placed on this GM as its available memory could be distributed among multiple LCs (e.g. 4 LCs with each 1GB of RAM). Consequently, a *list of candidate GMs* is provided by the dispatching policies. Based on this list, a linear search is performed by issuing VM placement requests to the GMs.

At the GM level, the actual VM scheduling decisions are taken. Therefore, four types of scheduling policies exist: *placement, overload relocation, underload relocation, and finally consolidation*. Placement policies (e.g. round-robin or first-fit) are triggered event-based to place incoming VMs on LCs. Similarly, relocation policies are called when overload (resp. underload) events arrive from LCs and aims at moving VMs away from heavily (resp. lightly loaded) nodes. For example, in case of overload situation VMs must be relocated to a more lightly loaded node in order to mitigate performance degradation. Contrary, in case of underload, for energy saving reasons it is beneficial to move VMs to moderately loaded LCs in order to create enough idle-time to transition the underutilized LCs into a lower power state (e.g. suspend).

Complementary to the event-based placement and relocation policies, consolidation policies can be specified which will be called periodically according to the system administrator specified interval to further optimize the VM placement of moderately loaded nodes. For example, a VM consolidation policy can be enabled to weekly optimize the VM placement by packing VMs on as few nodes as possible.

3.7 VM Relocation

The Snooze VM overload relocation policy is shown in Algorithm 1. It takes as input the overloaded LC along with its associated VMs and a list of LCs managed by the GM. The algorithm outputs a Migration Plan (MP) which specifies the new VM locations.

The overload relocation policy first estimates the LC utilization, computes the maximum allowed LC utilization, and the overloaded capacity delta (i.e. difference between estimated and maximum allowed LC utilization). Afterwards it gets the VMs assigned to the overloaded LC, sorts them in increasing order based on estimated utilization and computes a list of candidate VMs to be migrated. The routine to compute the migration candidates first attempts to find the most loaded VM among the assigned ones whose estimated utilization equals or is above the overloaded capacity delta. This way a single migration will suffice to move the LC out of overload state. Otherwise, if no such VM exists, it starts adding VMs to the list of migration candidates starting from the least loaded one until the sum of the estimated resource utilizations equals or is above the overload capacity delta. Finally the destination LCs are sorted in increasing order based on estimated utilization and migration candidates are assigned to them starting from the first one if enough capacity is available. Moreover, the new VM to LC mappings are added to the MP.

Algorithm 1 VM Overload Relocation

1: **Input:** Overloaded **LC** with the associated VMs and resource utilization vectors **UC**, list of destination **LCs**
2: **Output:** Migration Plan MP
3: **c** ← Estimate LC utilization
4: **m** ← Compute max allowed LC utilization
5: **o** ← Compute the amount of overloaded capacity (**c**, **m**)
6: VMs_{source} ← Get VMs from LC
7: Sort VM_{source} in increasing order
8: $VM_{candidates}$ ← computeMigrationCandidates(VMs_{source}, **o**)
9: Sort destination LCs in increasing order
10: **for all** $v \in VM_{candidates}$ **do**
11: LC_{fit} ← Find LC with enough capacity to host v (v, LCs)
12: **if** $LC_{fit} = \emptyset$ **then**
13: continue;
14: **end if**
15: Add (v, LC_{fit}) mapping to the migration plan
16: **end for**
17: **return** Migration plan MP

The underload relocation policy is depicted in Algorithm 2. It takes as input the underloaded LC and its associated VMs along with the list of LCs managed by the GM. It first retrieves the VMs from the underloaded LC and sorts them in decreasing order based on the estimated utilization. Similarly, LCs are sorted in decreasing order based on the estimated utilization. Then, VMs are assigned to LCs with enough spare capacity and added to the MP. The algorithms follows an *all-or-nothing approach* in which either all or none of the VMs are migrated. *Migrating a subset of VMs does not contribute to the energy saving objective (i.e. create idle times) and thus is avoided.* In order to avoid a ping-pong effect in which VMs are migrated back and forth between LCs, LCs are transitioned into a lower power state (e.g. suspend) once all VMs have been migrated thus they can not be considered as destination LCs during subsequent underload events.

3.8 VM Consolidation

VM consolidation is a variant of the multi-dimensional bin-packing problem which is known to be NP-hard. *Our system is not limited to any particular consolidation algorithm.* However, because of the NP-hard nature of the problem and the need to compute solutions in a reasonable amount of time it currently implements a simple yet efficient two-objective (i.e. *minimizes the number of*

Algorithm 2 VM Underload Relocation

1: **Input:** Underloaded **LC** with the associated VMs and resource utilization vectors **UC**, list of destination **LCs**
2: **Output:** Migration Plan **MP**
3: $VM_{candidates} \leftarrow$ Get VMs from underloaded LC
4: Sort $VM_{candidates}$ in decreasing order
5: Sort LCs in decreasing order
6: **for all** $v \in VM_{candidates}$ **do**
7: $LC_{fit} \leftarrow$ Find LC with enough capacity to host v
8: **if** $LC_{fit} = \emptyset$ **then**
9: Clear migration plan
10: break;
11: **end if**
12: Add (v, LC_{fit}) mapping to the migration plan
13: **end for**
14: **return** Migration plan **MP**

LCs and migrations) polynomial time greedy consolidation algorithm. Particularly, a modified version of the Sercon [7] algorithm is integrated which differs in its termination criteria and the number of VMs which are removed in case not all VMs could be migrated from a LC. Sercon follows an all-or-nothing approach and attempts to move VMs from the least loaded LC to a non-empty LC with enough spare capacity. Either all VMs can be migrated from a host or none of them will be. Migrating only a subset of VMs does not yield to less number of LCs and thus is avoided.

The pseudocode of the modified algorithm is shown in Algorithm 3. It takes as input the LCs including their associated VMs. LCs are first sorted in decreasing order based on their estimated utilization. Afterwards, VMs from the least loaded LC are sorted in decreasing order, placed on the LCs starting from the most loaded one and added to the migration plan. If all VMs could be placed the algorithm increments the number of released nodes and continues with the next LC. Otherwise, all placed VMs are removed from the LC and MP and the procedure is repeated with the next loaded LC. The algorithm terminates when it has reached the most loaded LC and outputs the MP, number of used nodes, and number of released nodes.

3.9 Migration Plan Enforcement

VM relocation and consolidation algorithms output a migration plan (MP) which specifies new mapping of VMs to LCs required to transition the system from its current state to the new optimized one. Migration plan is enforced only if it *yields to less LCs*. Enforcing the migration plan computed by the relocation and consolidation algorithms of our framework is straightforward as it only involves moving VMs from their current location to the given one. Note that, unlike other works (e.g. [9]) our algorithms do not introduce any sequential dependencies or cycles. Particularly, *VMs are migrated to an LC if and only if enough capacity is available on it without requiring other VMs to be moved away first.*

Migrations can happen either sequentially or in parallel. In the former case only one VM is moved from the source to the destination LC at a time, while the latter allows multiple VMs to be migrated concurrently. Given that modern hypervisors (e.g. KVM) support parallel migrations there is no reason not to do so given that enough network capacity is available. This is exactly what our system does.

Algorithm 3 VM Consolidation

```
1: Input: List of LCs with their associated VMs and resource utilization vectors UC
2: Output: Migration Plan MP, nUsedNodes, nReleasedNodes
3: MP ← ∅
4: nUsedNodes ← 0
5: nReleasedNodes ← 0
6: leastLoadedControllerIndex ← |LCs| − 1
7: while true do
8:    if leastLoadedControllerIndex = 0 then
9:       break;
10:   end if
11:   Sort LCs in decreasing order
12:   LC_least ← Get the least loaded LC (leastLoadedControllerIndex)
13:   VMs_least ← Get VMs from LC_least
14:   if VMs_least = ∅ then
15:      leastLoadedControllerIndex ← leastLoadedControllerIndex - 1
16:      continue;
17:   end if
18:   Sort VMs_least in decreasing order
19:   nPlacedVMs ← 0
20:   for all v ∈ VMs_least do
21:      Find suitable LC to host v
22:      if LC = ∅ then
23:         continue;
24:      end if
25:      LC_least ← LC_least ∪ {v}
26:      MP ← MP ∪{v}
27:      nPlacedVMs ← nPlacedVMs + 1
28:   end for
29:   if nPlacedVMs = |VMs_least| then
30:      nReleasedNodes ← nReleasedNodes + 1
31:   else
32:      LC_least ← LC_least \ VMs_least
33:      MP ← MP \VMs_least
34:   end if
35:   leastLoadedControllerIndex ← leastLoadedControllerIndex - 1
36: end while
37: nUsedNodes ← |LCs| - nReleasedNodes
38: return Migration plan MP, nUsedNodes, nReleasedNodes
```

Still, there exists a caveat here related to the pre-copy live migration termination criteria of the underlying hypervisor. For example, in KVM live migration can last forever (i.e. make no progress) if the number of pages that got dirty is larger than the number of pages that got transferred to the destination LC during the last transfer period.

In order to detect and resolve such situations Snooze spawns a *watchdog* for each migration. Watchdog enforces convergence after a system administrator predefined convergence timeout given the migration is still pending. Therefore it *suspends the VM* thus preventing further page modifications. The hypervisor is then able to finish the migration and restart the VM on the destination LC.

3.10 Power Management

In order to conserve energy, idle nodes need to be transitioned into a lower power state (e.g. suspend) after the migration plan enforcement. Therefore, Snooze integrates a power management module, which can be enabled by the system administrator to periodically observe the LC utilization and trigger power-saving state transitions (e.g. from active to suspend) once they become idle (i.e. do not host any VMs).

Particularly, power management works as follows. Snooze can be configured to keep a number of reserved LCs always on in order to stay reactive during periods of low utilization. Other LCs are automatically transitioned into a lower power state after a predefined idle time threshold has been reached (e.g. 180 sec) and marked as passive. Passive resources are woken up by the GMs either upon

new VM submission or overload situation when not enough active capacity is available to accommodate the VMs. Therefore a wakeup threshold exists which specifies the amount of time a GM will wait until the LCs are considered active before starting another placement attempt on those LCs.

The following power saving actions can be enabled if hardware support is available: shutdown, suspend to ram, disk, or both. Thereby, different shutdown and suspend drivers can be easily plugged in to support any power management tools. For example, shutdown can be implemented using IPMItool or by simply calling the Linux native shutdown executable.

Finally to enable LC power on, wakeup drivers can be specified. Currently, two wakeup mechanisms are supported in Snooze: IPMI and Wake-On-Lan (WOL).

4 Evaluation

4.1 System Setup

Snooze was deployed on *34 power metered HP ProLiant DL165 G7 nodes* of the Grid'5000 experimental testbed in Rennes (France) with one EP, one GL, one GM and 31 LCs. All nodes are equipped with two AMD Opteron 6164 HE CPUs each having 12 cores (in total 744 compute cores), 48 GB of RAM, and a Gigabit Ethernet connection. They are powered by six APC AP7921 power distribution units (PDUs). Power consumption measurements and the benchmarking software execution are done from two additional Sun Fire X2270 nodes in order to avoid influencing the measurement results.

The node operating system is Debian with a 2.6.32-5-amd64 kernel. All tests were run in a homogeneous environment with qemu-kvm 0.14.1 and libvirt 0.9.6-2 installed on the machines. Each VM is using a QCOW2 disk image with the corresponding backing image hosted on a Network File System (NFS). Debian is installed on the backing image and uses a ramdisk in order to speed up the boot process. The NFS server is running on the EP with its directory being exported to all LCs. VMs are configured with 6 VCORES, 4GB RAM and 100 MBit/sec network connection. Note that libvirt currently does not provide any means to specify the network capacity requirements. Therefore, Snooze wraps around the libvirt template and adds the necessary network capacity (i.e. Rx and Tx) fields.

Tables 1, 2, 3, and 4 show the system settings used in the experiments.

Table 1: Thresholds

Resource	**MIN, MID, MAX**
CPU,	0.2, 0.9, 1
Memory	0.2, 0.9, 1
Network	0.2, 0.9, 1

Table 2: Estimator

Parameter	**Value**
Packing density	0.9
Monitoring backlog	15
Resource estimators	average
Consolidation interval	10 min

Table 3: Scheduler

Policy	Algorithm
Dispatching	RoundRobin
Placement	FirstFit
Overload	see 3.7
Underload	see 3.7
Consolidation	see 3.8

Table 4: Power Management

Parameter	Value
Idle time threshold	2 min
Wakeup threshold	3 min
Power saving action	shutdown
Shutdown driver	system
Wakeup driver	IPMI

4.2 Experiment Setup

Our study is focused on evaluating the energy and performance benefits of the Snooze energy-saving mechanisms for dynamic web workloads. To make the study realistic, the experiment is set up in a way that reflects a *real-world web application deployment:* An extensible pool of VMs, each hosting a copy of a backend web application running on a HTTP server, while a load-balancer accepts requests coming from an HTTP load injector client (see Figure 2). Both the load-balancer and load injector are running on the Sun Fire X2270 nodes.

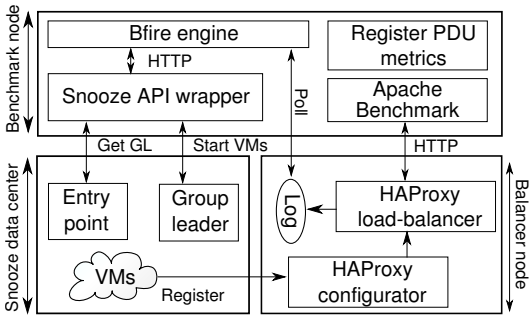

Figure 2: Experiment Setup

The backend application consists of a single HTTP endpoint, which triggers a call to the *stress* tool [14] upon each request received. Each stress test loads all VM cores during one second and uses 512 MB of RAM.

The load-balancer tool used is HAProxy v1.4.8, which is a *state-of-the-art load-balancer used in large-scale deployments* [15]. HAProxy is configured in HTTP mode, four concurrent connections maximum per backend, round-robin algorithm, and a large server timeout to avoid failed requests.

Finally, the load injector tool is the well-known Apache benchmark tool [16]. It is configured to simulate 20 concurrent users sending a total number of 15000 requests. According to our experiments these parameters provide the best trade-off between the experiment execution time and the effectiveness of illustrating the framework features.

The initial deployment configuration of the backend VMs is done using the Bfire tool [17], which provides a domain-specific language (DSL) for declaratively describing the provisioning and configuration of VMs on a cloud provider. Bfire also allows the monitoring of any metric and provides a way to describe elasticity rules, which can trigger up- or down-scaling of a pool of VMs when a key

performance indicator (KPI) is below or over a specific threshold. This tool is currently developed by INRIA within the BonFIRE project [18]. A thin wrapper was developed to make Snooze Bfire-compatible (i.e. interact with the Snooze RESTful API to provision VMs).

The experiment lifecycle is as follows: our Bfire DSL is fed into the Bfire engine, which initially provisions one backend VM on one of the physical nodes. At boot time, the backend VM will automatically register with the load-balancer so that it knows that this backend VM is alive. Once this initial deployment configuration is ready, the Bfire engine will start the Apache benchmark against the load-balancer. During the whole duration of the experiment, Bfire will also monitor in a background thread the time requests spent waiting in queue at the load-balancer level (i.e. before being served by a backend application). Over time, this KPI will vary according to the number of backend VMs being available to serve the requests. In our experiment, if the *average value of the last 3 acquisitions of that metric is over 600ms* (an acceptable time for a client to wait for a request), then a scale-up event will be generated, which will increase the backend pool by *four new VMs at once*. If the KPI is below the threshold, then nothing happens. This elasticity rule is monitored every 15 seconds, and all newly created VMs must be up and running before it is monitored again (to avoid bursting). Meanwhile, an additional background process is registering the power consumption values coming from the PDUs to which the physical nodes are attached.

At the end of the experiment, we show the performance (i.e. response time) of the application, the power consumption of the nodes, the number of VMs and live migrations. Moreover, we visualize all the events (i.e. Bfire, relocation, consolidation, power management) which were triggered in our system during the experiments.

Two scenarios are evaluated: (1) No energy savings, to serve as a baseline; (2) Energy savings enabled (i.e. underload relocation, consolidation and power management). In both scenarios overload detection is enabled.

4.3 Elastic VM Provisioner Events

The elastic VM provisioner (i.e. Bfire) events (i.e. READY, SCALING, and SCALED) without and with energy savings enabled (red resp. green colored) are shown in Figure 3. The experiment starts by provisioning one backend VM which results in the provisioner to become READY. When it becomes ready we start the actual benchmark which soon saturates the VM capacity. Bfire reacts by SCALING up the number of VMs to four. It takes approximately five minutes to provision the VMs. This is reflected in the subsequent SCALED event which signals the VM provisioning success. The same process happens until the end of the benchmark execution. In total four SCALING (resp. SCALED) are triggered which result in 17 VMs to be provisioned by the end of the Apache benchmark. Note that the *experiment with energy savings enabled lasts a bit more (1.2% of time) than without energy savings* because of the need to power on nodes and lightly increased response time (see the following paragraphs).

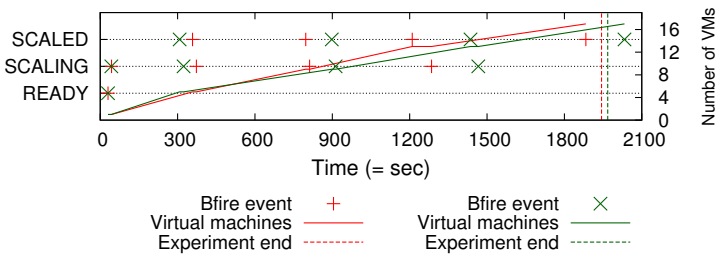

Figure 3: Elastic VM provisioner events

4.4 Apache Benchmark Performance

The Apache benchmark results (i.e. response time for each request) are depicted in Figure 4. As it can be observed, response time increases with the number of requests in both cases (i.e. without and with energy savings). However, more interestingly is the fact that response time is *not significantly impacted when energy savings are enabled*. Particularly, in both scenarios a response time peek exists at the beginning of the experiment. Indeed, one backend VM is quickly saturated. However, when times passes only minor performance degradation can be observed.

The main reason for the minor performance degradation lies in the fact that once energy savings are enabled, *servers are powered down*, thus increasing the time requests remain in the HAProxy queue until they can be served by one of the backends. Moreover, Bfire dynamically increases the number of VMs with growing load. Increasing the number of VMs involves scheduling, powering on LCs as well as a software provisioning phase in which tools are installed on the scheduled VMs in order to register with HAProxy. This requires time and thus impacts application performance (i.e. requests are queued). Performance could be further improved by taking proactive scaling up decisions. Finally, underload relocation and consolidation are performed which involve VM migration which contributes to the performance degradation.

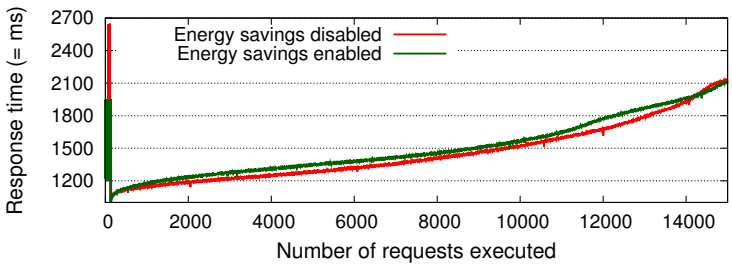

Figure 4: Apache benchmark performance

4.5 System Power Consumption and Events

The system power consumption without and with energy savings is depicted in Figure 5.

Without energy savings our experimental data center first consumes approximately *5.7 kW of idle power*. With the start of the benchmark the load increases to *6.1 kW* and falls back with the end of the evaluation. Note that our experiments did not fully stress all the 744 compute cores which would have resulted in even higher power consumption ($\sim$ 7.1 kW) but would also have made harder to conduct the experiment due to the increased execution time.

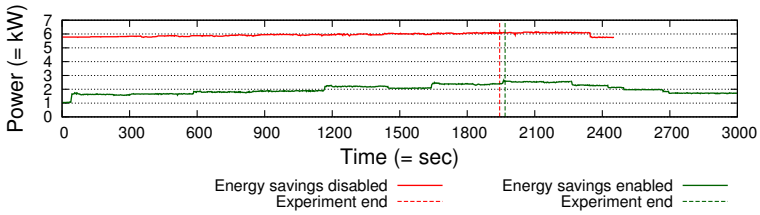

Figure 5: Power consumption

Snooze overcommits nodes by allowing them to host more VMs than physical capacity allows it. This leads to overloaded situations requiring VMs to be live migrated. In this context we distinguish between two types of events: overload relocation (OR) and migration plan enforced (MPE). The former is triggered in case of overload situation and results in a migration plan which needs to be enforced. MPE events signal the end of the enforcement procedure. Figure 6 shows the event profile including the number of migrations. As it can be observed the first two OR events trigger five migrations. This is due to the fact that the First-Fit placement is performed upon initial VM submission. This leads to an overload situation on the LCs which needs to be resolved. However, as time progresses the number of migrations decreases as VMs are placed on more lightly loaded LCs.

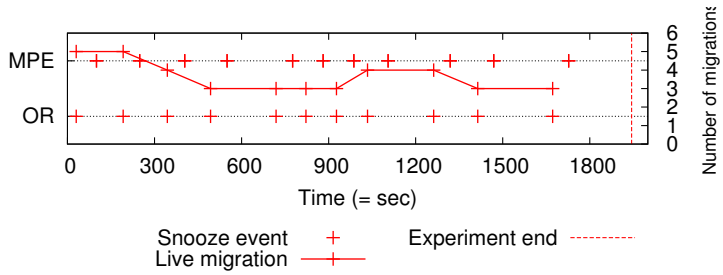

Figure 6: Snooze system events without energy savings

With energy savings enabled, when the experiment starts the system is idle, and thus the nodes have been powered down by Snooze, reducing the power consumption to approximately one kW (see Figure 5).

When the benchmark is started the system reacts by taking actions required to provision just as many nodes as needed to host the VMs. This results in the power consumption following the system utilization (i.e. increasing number of VMs). Note that the power consumption never drops to the initial value (i.e. one kW) as VMs are kept in the system in order to illustrate the framework mechanisms. Consequently, once idle they still consume additional power.

In a production environment VMs would be shutdown by the customers thus resulting in additional power savings.

Particularly, the following actions presented in Figure 7 are performed: (1) detect LC underload and overload; (2) trigger underload and overload relocation (UR resp. OR) algorithms; (3) enforce migration plans (MPE); (4) perform periodic consolidation (C); (5) take power saving actions such as power up and down (PUP resp. PDOWN) depending on the current load conditions. In order to get an insight in the system behaviour we have captured all these events.

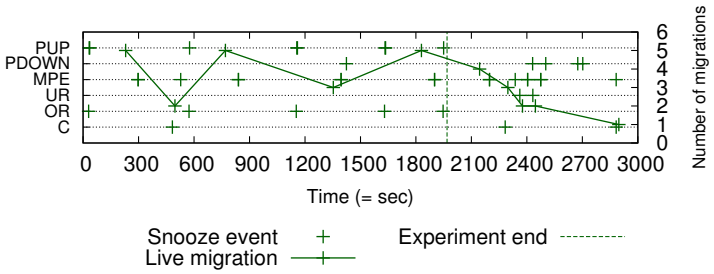

Figure 7: Snooze system events with energy savings enabled

During the benchmark execution the first OR event appears as the system becomes overloaded. The overload situation is resolved by powering up one LC and migrating five VMs. Then consolidation is started which migrates two VMs. The system continues to react to OR/UR events and adapt the data center size according to the current load (i.e. PUP and PDOWN events follow) until the end of the benchmark. Note that the number of migrations decreases with the benchmark execution time as the HAProxy load decreases with increasing number of backend VMs thus resulting in less OR events. Towards the end of the benchmark UR happens and results in a series of PDOWN events. Finally, consolidation is started and improves the VM placement by migrating one VM. *This shows that relocation and consolidation are complementary.*

Putting all the results together, data center energy consumption measured during the benchmark execution without and with power management enabled amounted to 3.19 kWh (34 nodes), respectively 1.05 kWh (up to 11 nodes), resulting in *67% of energy being conserved*. We estimated that for the same workload with a smaller data center size of 17 nodes, the energy gains would have been approximately 34%.

5 Conclusions and Future Work

This paper has presented and evaluated the energy management mechanisms of a unique holistic energy-aware VM management framework called Snooze. Snooze has a direct practical application: it can be either utilized in order to efficiently manage production data centers or serve as a testbed for advanced energy-aware VM scheduling algorithms.

To the best of our knowledge this is the first cloud management system which integrates and experimentally evaluates most of the required mechanisms to dynamically reconfigure virtualized environments and conserve energy within

a holistic framework. Particularly, Snooze ships with integrated VM monitoring and live migration support. Moreover, it implements a resource (i.e. CPU, memory and network) utilization estimation engine, detects overload and underload situations and finally performs event-based VM relocation and periodic consolidation. Snooze is the first system implementing the Sercon consolidation algorithm which was previously only evaluated by simulation. Finally, once energy savings are enabled, idle servers are automatically transitioned into a lower power state (e.g. suspend) and woken up on demand.

The Snooze energy management mechanisms have been extensively evaluated using a realistic dynamic web deployment scenario on *34 power-metered nodes of the Grid'5000 experimental testbed*. Our results have shown that the system is able to dynamically scale the data center energy consumption proportionally to its utilization thus allowing it to conserve substantial power amounts with only limited impact on application performance. In our experiments we have shown that with a realistic workload up to *67% of energy could be conserved*. Obviously the achievable energy savings highly depend on the workload and the data center size.

In the future we intend to extend our work to scientific and data analysis applications and evaluate different power management actions (e.g. suspend to ram, disk, both). Moreover we plan to integrate our previously proposed nature-inspired VM consolidation algorithm [19] and compare its scalability with the existing greedy algorithm as well as alternative consolidation approaches (e.g. based on linear programming). In addition we plan to apply machine learning techniques in order to predict VM resource utilization peaks and trigger pro-active relocation and consolidation actions. Finally, power management features will be added to the group leader in order to support power cycling of idle group managers. Ultimately, Snooze will become an open-source project (`http://snooze.inria.fr`) in Spring 2012.

6 Acknowledgments

We would like to thank Oleg Sternberg (IBM Research Haifa), Piyush Harsh (INRIA), Roberto G. Cascella (INRIA), Yvon Jégou (INRIA), and Louis Rilling (ELSYS Design) for all the great feedbacks. This research is funded by the French *Agence Nationale de la Recherche (ANR)* project EcoGrappe under the contract number ANR-08-SEGI-000. 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 (see `https://www.grid5000.fr`).

References

[1] G. International, "Make IT Green: Cloud Computing and its Contribution to Climate Change," 2010, http://www.greenpeace.org/usa/en/media-center/reports/make-it-green-cloud-computing/. 1

[2] Rackspace, "Hosting reports third quarter," 2011. [Online]. Available: http://ir.rackspace.com/phoenix.zhtml?c=221673&p=irol-newsArticle&ID=1627224&highlight= 1

[3] J. Koomey, "Growth in data center electricity use 2005 to 2010," Oakland, CA, USA, August 2011. [Online]. Available: http://www.analyticspress.com/datacenters.html 1

[4] A. Kivity, "kvm: the Linux virtual machine monitor," in *OLS '07: The 2007 Ottawa Linux Symposium*, Jul. 2007. 1, 3.1

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003. 1, 3.1

[6] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, ser. NSDI'07, 2007. 1, 2

[7] A. Murtazaev and S. Oh, "Sercon: Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing," *IETE Technical Review*, vol. 28 (3), 2011. 1, 3.8

[8] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, May 2011. 1, 2

[9] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2009. 1, 2, 3.9

[10] E. Feller, L. Rilling, and C. Morin, "Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds," in *The 12th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid)*, May 2012. 1

[11] Í. Goiri, J. L. Berral, O. Fitó, F. Julià, R. Nou, J. Guitart, R. Gavalda, and J. Torres, "Energy-efficient and multifaceted resource management for profit-driven virtualized data centers," *Future Generation Computer Systems*, vol. Vol. 28 (5), 2012. 2

[12] P. Graubner, M. Schmidt, and B. Freisleben, "Energy-Efficient Management of Virtual Machines in Eucalyptus," in *Proceedings of the 4th IEEE International Conference on Cloud Computing (CLOUD)*, July 2011. 2

[13] VMware, "Distributed Resource Scheduler (DRS)," 2012. [Online]. Available: http://www.vmware.com/products/drs/ 2

[14] "Stress tool," 2012. [Online]. Available: http://weather.ou.edu/~apw/projects/stress/ 4.2

[15] "HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer." [Online]. Available: http://haproxy.1wt.eu/ 4.2

[16] "ab - Apache HTTP server benchmarking tool," 2012. [Online]. Available: http://httpd.apache.org/docs/2.0/programs/ab.html 4.2

[17] "Bfire - A powerful DSL to launch experiments on BonFIRE," 2012. [Online]. Available: https://github.com/crohr/bfire 4.2

[18] "BonFIRE - Testbeds for Internet of Services Experimentation," 2012. [Online]. Available: http://www.bonfire-project.eu/ 4.2

[19] E. Feller, L. Rilling, and C. Morin, "Energy-Aware Ant Colony Based Workload Placement in Clouds," in *Proceedings of the 12th IEEE/ACM International Conference on Grid Computing (GRID)*, September 2011. 5

Contents

1 Introduction **3**

2 Background **4**

3 Energy Management in IaaS Clouds: A Holistic Approach **5**
3.1 System Model and Assumptions 5
3.2 System Architecture 5
3.3 System Parameters 6
3.4 Resource Monitoring and Anomaly Detection 7
3.5 Resource Utilization Estimations 7
3.6 Energy-Aware VM Scheduling 8
3.7 VM Relocation 8
3.8 VM Consolidation 9
3.9 Migration Plan Enforcement 10
3.10 Power Management 11

4 Evaluation **12**
4.1 System Setup 12
4.2 Experiment Setup 13
4.3 Elastic VM Provisioner Events 14
4.4 Apache Benchmark Performance 15
4.5 System Power Consumption and Events 15

5 Conclusions and Future Work **17**

6 Acknowledgments **18**

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399