Hybrid approach for energy aware management of multi-cloud architecture integrating user machines

D Borgetto, Rodrigue Chakode, Benjamin Depardon, Cédric Eichler, Jean-Marie Garcia, H Hbaieb, Thierry Monteil, E Pelorce, A Rachdi, Ahmad Al Sheikh, et al.

Noname manuscript No.
(will be inserted by the editor)

Hybrid approach for energy aware management of multi-cloud architecture integrating user machines

D. Borgetto · R. Chakode · B. Depardon · C. Eichler · J.M. Garcia · H. Hbaieb · T. Monteil · E. Pelorce · A. Rachdi · A. Al Sheikh · P. Stolf

Received: date / Accepted: date

Abstract The arrival and development of remotely accessible services via the cloud has transfigured computer technology. However, its impact on personal computing remains limited to cloud-based applications. Meanwhile, acceptance and usage of telephony and smartphones have exploded. Their sparse administration needs and general user friendliness allows all people, regardless of technology literacy, to access, install and use a large variety of applications.

We propose in this paper a model and a platform to offer personal computing a simple and transparent usage similar to modern telephony. In this model, user machines are integrated within the classical cloud model, consequently expanding available resources and management targets.

In particular, we defined and implemented a modular architecture including resource managers at different levels that take into account energy and QoS concerns. We also propose simulation tools to design and size the underlying infrastructure to cope with the explosion of usage.

H. Hbaieb, E. Pelorce
Degetel, 46 Avenue du General Leclerc, 92100 Boulogne-Billancourt, France
E-mail: (hhbaieb, epelorce)@degetel.com

D. Borgetto, P. Stolf
IRIT; 118 Route de Narbonne, F-31062 Toulouse, France
Univ de Toulouse, UPS F-31400, UT2J, F-31100 Toulouse, France
E-mail: (borgetto, stolf)@irit.fr

A. Rachdi, A Al Sheikh
6, Avenue Marcel Doret, 31500, Toulouse France
E-mail: rachdi, alsheikh@qosdesign.com

C. Eichler, T. Monteil, J.M. Garcia
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, INSA F-31400, F-31100 Toulouse, France
E-mail: (ceichler, monteil, jmg)@laas.fr

R. Chakode, B. Depardon
56, boulevard Niels Bohr CS 52132 69100 Villeurbanne, France
E-mail: (benjamin.depardon, rodrigue.chakode)@sysfera.com

Functionalities of the resulting platform are validated and demonstrated through various utilization scenarios. The internal scheduler managing resource usage is experimentally evaluated and compared with classical methodologies, showing a significant reduction of energy consumption with almost no QoS degradation.

Keywords hybrid cloud · sharing users resources · energy aware schedulers · autonomic computing

1 Introduction

Mobile phones and personal computers are rapidly converging. However, on one hand, the service/application access model on mobile phones has been designed to smooth this access to new users with free economic models, pay-as-you-go or one-use billing. On the other hand, for computers, we are still following an operational model where the user must manage his machine himself: configuring the operating system, buying software, installing and updating them with all the underlying consequences that may occur with a non-expert user. At work, help from a team of specialists can avoid this kind of situation, but requires a lot of human resources. This is favourable to the emergence of a new computing operational model, built on the achievements of mobile telephony like the new services and uses, the extensions to the access models for remote software resources like cloud computing and peer-to-peer.

The objective of this article is to build a hybrid working model, that takes benefit from a mix between local installation/management and remote access to software and services located in distant dedicated data centers or even in other non-used consumers machines. The work presented in this paper has been conducted under a French Research project named ANR SOP (think global Services for persOnal comPuter).

We propose to study, design and implement a complete prototype for handling a machine, an operating system and software package subscription. We present a platform composed of a resource management middleware, schedulers, a market software and an autonomic manager. In parallel, a simulator demonstrates the feasibility of this approach for several hundreds of thousands or even millions of users.

This paper is organized as follows: section 2 will describe related works. Section 3 introduces the SOP model and details the prototype architecture. Section 4 presents the meta-scheduler algorithm and an energy-aware algorithm. Then, section 5 presents a simulation tool used for dimensioning large scale cloud infrastructures. Next, section 6 details scenarios to validate the elaborated platform and analyses experiments made with the scheduler algorithm on a real platform. Finally, section 7 concludes the paper and covers the discussion part.

2 Background

After several stages of evolution of the Internet, the Information Age today sees a new way to access and use IT resources. These remote resources are intended to make the information processing transparent, as perceived by the user and also have the ability to be accessible regardless of their geo-location. The concept of cloud implements the services necessary for the use of remote resources [26–28]. Transparent use of cloud could be made through advanced interface like in [1].

Computing centers hosting cloud services are now composed of several thousands of machines in the same data center. They see their growing use, due to the performance of resources, services and transparency in the use and operation they offer [29].There have been extensive studies on how to manage data centers and clouds in regards to the power consumption and QoS trade-off. Especially in the VM allocation and cloud reconfiguration area.

The architectural framework for energy efficient clouds proposed in [13] introduces notions of VM provisioning, as well as allocation and reallocation of VMs on physical machines. The authors use the CPU resource to dynamically consolidate the VMs, thus reducing significantly the global energy consumption of the infrastructure. Moreover, the authors implement a VM replacement policy based on a double threshold (low and high), as well as a migration minimization policy, in order to limit the number of migrations, for instance from an overloaded host. The framework is evaluated according to two different metrics: total energy consumption of the infrastructure, and service level agreement (SLA) violation percentage. The results show energy savings going up to 66% compared to an approach without migrations, while keeping a low rate of SLA violations. They study the impact of the Service Level Agreement degradation on energy consumption reduction.

Xiao et al. in [24] present a system based on application requirement to support green computing in the cloud. The main way to do so is by consolidation. They introduce the concept of skewness, which represents the imbalance over the different resources consumption of a server, applied in their case to the CPU, memory and disk. Reducing this imbalance is aimed at increasing individual host's utilization. The consolidation algorithm they present is based on a multi-threshold approach to define cold and hot spots. It then reallocates VMs to reduce cold spots and mitigate hot spots. The authors also combine this approach to a virtual machine load prediction to better provision the resources.

Based on the observation that the type of application running inside a virtual machine can drastically change its behaviour depending on where the VM image is, Yang et al. in [25] propose different workload characteristic-aware bin packing algorithms to be applied for consolidation in cloud environment. They propose in their paper a dynamic programming optimal algorithm and an approximation heuristic that they compare between each other, the performance metric being the number of hosts used. The approximation algorithm is a bin packing algorithm called SEP-pack, which stands for separate packing. The idea being that, since it is possible to characterize the different VMs according to their type of workload, data intensive or CPU intensive, they have to be treated differently. That way, the heuristics allocate separately the VMs by separating the bins in two, allocating the data intensive VMs first in one part of the bins, and the rest in what is left. They evaluate the algorithms by comparing the near optimal solution given by SEP-pack to the optimal solution given by dynamic programming. The number of bins used by SEP-pack is obviously greater than the optimal solution, but is rather close to it, depending on how the separation of the bins is made. Characterization of job could also use trace [22].

In [16], the authors present two algorithms for energy consumption reduction and migration cost mitigation in the cloud. They compute an exact solution based on integer linear programming in order to minimize the energy consumption under constraints of CPU, memory and storage of the VMs. Their approach allows to assign a power budget to each host, that should not be overran. They compare it with an energy-aware best fit heuristic. The evaluation is made between the optimal allocation provided by a linear solver, and an energy aware best fit allocation heuristic defined to find a sub optimal solution to the problem. The evaluation by simulation shows that although the ILP approach gives optimal placement, it does so in reasonable time most of the time for data center with under of 400 hosts and less than 500 VMs, the best fit heuristic being usable to compute faster solutions.

Another approach to the virtual machine packing problem by integer linear programming is also presented in [21]. The authors have worked on different VM packing algorithms that aim at reducing the energy consumption, modeled as the cost of the hosts' electrical consumption and the cost of the reconfiguration. That way, the authors aim at reducing power consumption while maintaining a certain level of performance, and diminishing the cost of reconfiguration by live migration. Reducing collisions of VM live migration is made by putting a hard constraint on the number of VMs that can migrate from and to a host. Takahashi et al. then define two algorithms, a matching-based algorithm based on graph and the modeling of the problem, that computes the optimal solution, and a greedy heuristic that computes the solution faster. The algorithms are then compared in terms of power consumption, performance degradation, and computation time, by numerical simulation using traces of the T2K-Tsukuba supercomputer. They show that there is a need for a relaxation of the problem regarding the allowed number of migrations to be able to compute solutions even if the system is oversubscribed. The results of the experiments show that the algorithms yield an energy reduction of between 15% and 50%, and that if the greedy algorithm has the advantage in terms of power consumption, the matching-based algorithm has the advantage in terms of performance degradation.

All those approaches are akin to ours, as the main focus is to save energy by powering off hosts unloaded by consolidation. However, small differences are to be seen on the sort of information the manager has, whether it is categorizing the virtual machines, being able to oversubscribe the hosts, and how to manage virtual machines migration and the bottleneck induced. The main difference being the ability to take into account the actual overhead of powering on and off hosts.

3 New Model Proposed

3.1 Introduction

In this hybrid model called SOP model (for thinking global Service for persOnal comPuter), users connect their terminal and choose the applications they want to access. For some of which they will have to pay for the related licenses, whereas others will be available for free. After this selection phase, there will be either a software installation on the local machine, or a remote access to the software. The choice of these setup policies is based on the user, the machine (not powerful enough), the resource center (load, energy consumption, licenses), the software (not accessible remotely, bandwidth hog, multi-user), the desired quality of service (speed, privacy, redundancy), the energy consumption, etc. Once the installation is finished, the software will be available transparently for the user with either: a remote access to a computer center, a local use, or even an access to unused resources of another individual for energy conservation purposes. The overall software architecture, including the operating systems is deployed, managed and optimized by the service provider in a fully transparent manner and dynamic way.

3.2 Proposed architecture

The proposed architecture involves several softwares and new policies to manage this complex system:

- a well known cloud software (OpenNebula) to manage several clouds with:
  - a classical data center used as cloud platform for Infrastructure As A Service (called in figure 1 Cloud provider)
  - clouds made with the users computers (called in figure 1 User Cloud)
- Extension of cloud resource management policies of OpenNebula (SOPVP) to manage this complex architecture with energy and Quality of Service constraints
- A software to federate the different clouds: Vishnu and a new meta-scheduler to manage several clouds with energy and Quality of Service constraints
- A market place to help non-expert user to access computer capabilities with a ”green attitude”
- An autonomic manager to provide transparent management of this highly distributed architecture

We consider also a set of users that submit jobs for execution on the servers of a data center. Each individual user can be either idle (inter-session phase)

Fig. 1 SOP Architecture

or involved in an on-going session of a remote application. We shall distinguish three broad categories of applications:

- Interactive: Applications in this class require resources for a finite duration which could be interspersed in time with periods of inactivity. OpenOffice (Document edition) is an example of a software application in this class.
- CPU-intensive: This class of applications are those that require physical resources for a continuous and finite duration of time. Scilab, for example, fits within this class.
- Permanent: This class contains software applications such as web servers or database servers that have no end date. Once a virtual machine hosting a web server is instantiated, it is assumed to be functional forever, waiting for requests or processing them simultaneously.

3.3 Vishnu

Vishnu[2] is an open-source distributed resource management middleware enabling to federate and manage distributed HPC clusters and cloud infrastructures, in order to offer to users a unified and easy-to-use user interface. Providing command line interfaces and application programming interfaces (API) in several programming languages, Vishnu provides users with a simple way to access and use the underlying resources, without needing any specific knowledges on the backend cluster/cloud resource managers. SLURM[6], LoadLeveler[7], OpenNebula[4], to name but a few, are supported resource manager backends. Figure 2 presents the global architecture of Vishnu.

Fig. 2 Vishnu Architecture

Within the SOP architecture, Vishnu acts as the main resource manager in front of the backend OpenNebula-based clouds. In particular, Vishnu responses to access requests by starting applications within VMs it deploys on appropriate resources. Vishnu has been modified to overcome the specificities of the SOP project: an increased support for OpenNebula, a new communication layer based on ZeroMQ [5] integration of SOP's meta-scheduler to distribute the jobs... When a user requests an application through the Market (see section 3.4), the request is forwarded to Vishnu which is responsible for handling the submission and the execution on the underlying clouds. To achieve that, Vishnu relies on the 4.1 algorithm to select the target cloud based on the underlying cloud accounting information. Once the cloud is selected, Vishnu relies on the target OpenNebula instance to execute the job. Basing on request parameters, including the target application and resource requirements (CPU, memory. . .), Vishnu is also in charge of preparing and requesting the creation of the related virtual machines to OpenNebula, and executing the requested application on these virtual machines. This phase of configuration and deployment should be automatized for easy use of cloud [15]. Vishnu then monitors the whole execution of each request, it controls the state of the applications during their execution, and once a request is completed, the virtual machine is destroyed to free up resources. The destruction of the virtual machine is orchestrated by Vishnu, but handled by OpenNebula.

3.4 Market

The application store is the main interface for SOP users. It offers a catalogue of available applications and an interactive help module allowing users to choose the tools they would like to buy. After the purchase, programs will be installed automatically. The solution is based on the open source content management system "Drupal".

The Market integrates an ecopoints allocation system in order to reward users whose machines have been used to launch a third-party application. It analyzes the job submission logs and OpenNebula logs and affects ecopoints to these users (proportionally to the VM runtime). All ecopoints can be used to buy applications on the market.

3.5 Scheduler

OpenNebula already provides a base scheduler with its installation. It is a configurable scheduler called *match-making scheduler* or *mm_sched* and is described in details in [10].

The *mm_sched* algorithm operates in three simple steps:

- Filter hosts that lack the required characteristics (e.g: that are not in the right state or don't have enough resources).
- Sort all hosts according to the ranking policy (described hereafter).
- The hosts with the highest rank are chosen for the allocation.

In this article we will present a SOPVP algorithm implemented in OpenNebula, and presented in Section 4.2.

3.6 Autonomic manager

The FrameSelf [18] architecture is based on the IBM autonomic manager architecture reference [23]. It is organized into four main modules which are Monitor, Analyzer, Planer and Executer. These modules share a same knowledge and dynamically enable the management of legacy entities using sensors (information from legacy) and effectors (action on legacy).

Knowledge base must be simple and generic to be easily handled by the four control loop modules. Knowledge elements generated during the management steps such as symptoms, requests for change, and change plans occurrences must be presented in generic formats independents from the managed resources. The resource identification and the description of its interactions must be represented using powerful representation tools. Frameself allows to use different models of knowledge like logical rules, semantic with inference rules or mathematical model like queuing formula. FrameSelf global architecture is described in Figure 3.

Fig. 3 FrameSelf architecture

The monitor module provides the mechanism that correlates events collected from a managed resource sensors and generates symptoms or reports. It receives as input events collected from managed entities and checks if symptom occurrences should be created as a response. The monitor module is decomposed into several subcomponents: normalizier, filter, aggregator, collector, etc.

The analyzer module provides the mechanism that correlates and models complex situations. These mechanisms allow the autonomic manager to learn about the environment and help predict situations. The analyzer receives as input symptom occurrences and checks if a request for change should be created as a response. The Analyzer is a composite module that contains a symptom collector, a request correlation and a request generator.

The planner provides the mechanisms that construct the action needed to achieve objectives. It exploits policies based on goal and environment awareness to guide its work. It receives as input symptom occurrences and generates action plans as response. The planner contains a plan generator and a policy manager.

The executor provides the mechanisms that control the execution of a plan with consideration for dynamic updates. It receives as input a sequence of actions and performs it by using managed entities effectors. The executor is a composite that contains a plans collector, an actions correlation and an action executor component able to execute script and action using different kind of technology (Web services, Rest, RMI, script file, etc.)

4 Schedulers

4.1 Meta-scheduler

The meta-scheduler handles the inter-cloud direction and redirection of VMs. As such, it has to comply to several requirements in order to maintain a coherent choice of the most suitable clouds. Since there is no knowledge of the actual state of the user clouds, the success rate, thus the quality, of a meta scheduler will be defined as its tendency to succeed, or not, the task placement. A VM can be sent to a user cloud without any guarantee that the cloud scheduler will be able to properly place it. In such a case, the VM will be sent away from the user cloud, to another one.

The meta scheduler will need to prevent sent away VMs to be redirected again to the same user cloud. At the same time, the meta scheduler will have to avoid starvation of the virtual machines by ensuring that a VM does not get bounced away too often, for example by ensuring that a VM is sent back to the data center cloud when it has been rejected too often. The choice of the data center cloud will be made because of its superior material capacity, compared to user clouds, that comprise only desktop and mobile computers, as opposed to powerful servers. We will thus choose a maximum number of times that a VM is allowed to go through the meta scheduler. The meta scheduler will have to maintain a cache of the allocation informations of the previous decisions.

In order to have a consistent behaviour, and because of the limited knowledge of the actual state of the different clouds, the meta scheduler will make its choice based on aggregated metrics. It will have to prioritize certain tendencies over others.

The decision algorithm is based on the following aggregated metrics, as well as generic informations like task id and type, as well as originating cloud if there is one and cloud owning the task, that has to be given for the meta scheduler to keep track of its choice. The aggregated metrics that have to be retrieved for the clouds are the global CPU and memory load, which is the percentage of CPU and MEM load of the entire cloud. The number of VMs and physical hosts are also to be sent to the meta scheduler. We will also send the number of hosts whose CPU load is over 50%, as well as the number of hosts whose MEM load is over 50%. Finally highest mean resource load has to be sent, which represent the maximum between the average CPU and average MEM load. Those metrics and their notations are summed up in table 1.

Table 1 Meta scheduler metrics and notations

c_id	ID of user cloud
$load_c^{CPU}$	Cloud's global CPU load
$load_c^{MEM}$	Cloud's global MEM load
$\#VM$	Number of VM in the cloud
$\#H$	Number of hosts in the cloud
$h_{<50\%}^{CPU}$	Number of hosts where CPU load is below 50%
$h_{<50\%}^{MEM}$	Number of hosts where MEM load is below 50%
$HML(CPU, MEM)$	Highest mean load between CPU and MEM

The first metric we are going to compute will be used to determine the portion of lightly loaded hosts. We arbitrarily defined that a lightly loaded host is represented by a resource load $load^r \leq 50\%$. The metric X and is defined as follow:

$$X = max(\frac{h^{CPU}_{<50\%}}{\#H}, \frac{h^{MEM}_{<50\%}}{\#H})$$

We then define the second metric which is the highest mean resource load of a cloud, represented by Y and defined as:

$$Y = HML(CPU, MEM)$$

The last metric is about the virtual machines running on a cloud. In order for this value to be comparable to others, we need to proceed in several steps. We have to first construct a sorted list of user clouds. The list will be sorted in increasing order according to the following comparison function:

$$ranking(c) = max(\frac{load_c^{CPU}}{\#VM}, \frac{load_c^{MEM}}{\#VM})$$

Once the clouds are sorted, we will use the value represented by the rank of each cloud in the list.

$$Z = rank(c)$$

We can now use the different values presented here to compute the choice metric to be maximized:

$$choice = \alpha X - \beta Y - \gamma Z$$

This metric is to be used in the algorithm presented in algorithm 1

The meta-scheduler algorithm has been implemented in Vishnu.

4.2 SOPVP

4.2.1 Problem Formulation

In this section we focus on the SOPVP algorithm which is the scheduler used in the data center. We first present the environment and the problem we are trying to solve. We are set in a cloud, operated over a single data center by a provider called in the figure1 Cloud Provider. The provider allows user to start and run VMs, with the constraint of having provided to them what they have required.

The cloud has H physical hosts, with the ability to power on and off when empty. Over time, VMs are run on the cloud for a certain duration. Over the duration T of the experiment, J VMs will be executed. Each VM has resource requirements, which we chose in our case to be CPU and RAM. Let's note vm^{CPU} the number of CPU required by a VM, and vm^{MEM} the number of MB of RAM required by the VM. The VMs will be allocated on the different hosts so that their requirements are matched. We will note $e_{vm,h} = 1$ if the VM vm is allocated to the host h, $e_{vm,h} = 0$ otherwise.

If h^{CPU} is the CPU capability and h^{MEM} the MEM capability of the host, the load of the host is defined as the following:

```
Data: Aggregated cloud monitoring values
Result: Cloud id $c_{id}$ of the chosen cloud
clouds[ ] C;
task t;
if task t originates from a cloud c in C then
    Send t to c;
end
if task t is not in cache then
    Add t to cache;
else
    Increase try counter in cache;
end
if number of tries exceeds MAX then
    Send c to data center;
end
Sort(C) according to the ranking function;
max_value = -infinity;
chosen_cloud = ∅;
foreach $c \in C$ do
    $X = max(\frac{h^{CPU}_{<50\%}}{\#H}, \frac{h^{MEM}_{<50\%}}{\#H})$;
    $Y = highest\_mean\_load(CPU, MEM)$;
    $Z = rank(c)$;
    $choice = X \times \frac{H}{2} - Y \times H - Z$;
    if $max\_value < choice$ then
        chosen_cloud = c;
        max_value = choice;
    end
end
return chosen_cloud;
```

Algorithm 1: Meta-scheduler algorithm

$$r \in R = \{CPU, MEM\} : \quad load^r_h = \sum_{vm \in J} \frac{vm^r \times e_{vm,h}}{h^r}$$

The load of the host over each resource considered by the VMs should never be exceeded.

Each VM is finite in time, thus will also be defined by its duration:

$$d_{vm} = t_{sched} + t_{boot} + t_{run}$$

Where t_{sched} comprises the scheduling time and the time for its implementation. The values t_{boot} and t_{run} represent respectively the time taken by the VM boot time, and the time during which the VM is in the *running* state. It also means that if a VM requires an hour of time on the cloud, it will effectively run the requested hour minus the time taken to spawn and boot the VM.

4.2.2 Metrics

We defined a hard bound on how the different jobs are to be allocated, meaning that a VM can't get less than required. This also means, that to be able to have a sort of QoS metric, we can't reach inside the VM to get response time for example, because this model is agnostic to what kind of application the VM is running. That's why we will define the QoS metric of the VMs run on the cloud by defining the *effective duration* of the VMs:

$$d_{vm}^{eff} = \frac{t_{run}}{d_{vm}}$$

An effective duration of 90% of a particular VM will mean that this VM spent 90% of the requested time in the running state.

The other performance metric we obviously want for our model is the energy consumption of the cloud. We will use the real energy consumption, calculated as the sum of all the power consumptions measured over all the hosts.

$$E = \int_0^t P_t dt = \int_0^t \sum_{h \in H} P_h dt$$

4.2.3 The algorithm

The implementation of the scheduler for the cloud that we defined will follow the same functioning pattern as *mm_sched*. We will have the same periodic calls to the scheduler in order to have a consistent comparison.

The scheduler has to take into account different constraints. The first is that it has to be separated into two distinct algorithms: one for the initial allocation of VMs, and one for the periodic reconfiguration of the VMs in the cloud, to better handle the subset of migrations that have to be done for energy reduction without having to reallocate all VMs. The algorithm allocates pending VMs based on a best-fit heuristic.

The algorithm to reallocate will bring the most energy savings by using consolidation to reduce the number of hosts used by the VMs when the system changed because some of the VMs have ended for instance. We used a modified vector packing algorithm, described as algorithm 2, to reallocate the virtual machines from one or several hosts to reduce the global host number. The approach used is based on the algorithm presented in [20], and is working by separating the hosts in as much list as we have dimensions for the allocation. Each list contains the hosts with more requirements in one particular resource. In our case, we will have two lists, one for the hosts that have more CPU than MEM, and one for the opposite. The CPU and MEM are not the same type of resource, since the CPU can be oversubscribed more easily than the memory. Furthermore, it is better not to provide an application with less memory than it requires. However, we took into account both resources at the same level to guarantee an execution time quality of service for applications running inside VMs.

Then, the VMs are allocated to the hosts, previously sorted, so that the allocation reduces the current resource imbalance. For example, if the VM has $vm^{CPU} > vm^{MEM}$, then we will pick preferably a host from the list where

$load_h^{MEM} > load_h^{CPU}$. That way, the imbalance of resource consumption will be reduced.

We have to choose the number of hosts that we will try to consolidate. That number can be chosen in several ways, like having thresholds to define low loaded hosts that will be chosen to consolidate, or, like we do in this paper, choosing to unload a fixed number of hosts. That number must vary regarding the global number of hosts and VMs. In addition, it will also have to depend on the number of concurrent migrations that can be made toward one host, and on the network infrastructure. Live migrations are putting a strain on the network as they transfer the memory pages of the VM potentially more than once, depending on how data intensive is the application. That being said, the number of hosts chosen for unloading needs to be capped to the maximum amount of concurrent migration one wants, as well as being tailored to match the infrastructure size.

We will compute the number of hosts to be 20% of the total host number H, with a minimum of one (1) host. We also chose to mitigate that number regarding the projected amount of migrations, as we will describe in 6.2.3. The algorithm 2 is presented for a number of hosts to unload equal to 1, since the extension to several hosts is rather straightforward.

As we can see, in the algorithm, one should note that the migrations are only enforced if and only if the host may be fully unloaded. It is an optimization choice. The aim is to avoid migrations that could be irrelevant because another decision may be better in the next scheduling loop and that will only generate overhead. The algorithms are evaluated in section 6.2.

5 Platform Design

Build and support a platform based on classical clouds and clouds built from customers hosts while ensuring service quality is challenging. One of the possibilities to anticipate problems, including design, is the use of simulation platforms. It is the objective of the NEST [3] environment to enable a provider to scale its platform based on its actual and forcasted usage.

As a matter of fact, NEST can model the cloud infrastructure represented in Fig.4 (Servers, LAN, Front-ends), the users accessing their favourite services and the network.

The simulation model is aimed at predicting response times perceived by users in a cloud computing environment under the SaaS model. We assume that each instance of a software application is executed within a virtual machine running on a computing node of a data center, and that VMs running concurrently on the same node share fairly its capacity. This model explicitly takes into account the different behaviours of the different classes of software applications (interactive, CPU-intensive or permanent).

Fig. 4 Nest simulation tool : inside a cloud provider platform

```
begin Consolidate VMs
    $H$ = sort(H, load ascending);
    $llh = h \in H$ where min($load_h^r$);
    $\{H^{CPU}, H^{MEM}, Migrations\} = \emptyset$;
    for $h \in H, h \neq llh$ do
        if $load_h^{CPU} > load_h^{MEM}$ then
            add $h$ to $H^{CPU}$;
        else
            add $h$ to $H^{MEM}$;
        end
    end
    success=True;
    foreach $vm$ : $(e_{vm,llh} = 1)$ do
        found = False ;
        if $vm^{CPU} > vm^{MEM}$ then
            found = Try to find a suitable host $h$ in $H^{MEM}$ first, then $H^{CPU}$;
        else
            found = Try to find a suitable host in $H^{CPU}$ first, then $H^{MEM}$;
        end
        if $found$ then
            add $(vm, h)$ to $Migrations$;
            change list of $h$ if needed ;
        else
            success=False;
        end
    end
    if $success=True$ then
        enforce $Migrations$;
    end
end
```

Algorithm 2: SOPVP Reallocation algorithm

Through SLA configurations for the services, the simulation tool allows analysing the performance of cloud infrastructures based on predefined user demands. In addition, it helps identifying bottlenecks that lead to poor performances, be it an overloaded server or an under-dimensioned infrastructure. Consequently, we can use this tool to dimension a cloud infrastructure in a "design by testing" approach. By defining traffic matrices for each of the services (user demands) and running some simulations, we can assume that the available infrastructure is able to handle the user demands if all user SLOs (Service Level Objectives) are satisfied. We then scale this demand until some QoS violations arise.

The simulation results contain metrics that guide the provider in the dimensioning process. These metrics include memory consumption and CPU utilisation at each node.

6 Scenario and validation

6.1 Scenario

We designed scenarios to validate the elaborated platform and demonstrate some of its functionalities. The three most relevant are described below. They are contiguous; the final state of a scenario is the initial state of the next one.

Fig. 5 Scenario for market place and remote execution

The first scenario, illustrated on the left-hand side of Fig. 5, depicts the *installation* of a new software application.

1. While browsing the store, the user 1 finds scilab, an application he is interested in. He consequently press a "download & install" button.
2. After handling any relevant budgetary actions, the application is either:
   – downloaded and installed on its machine. This process takes place if the application can be locally executed. The user can also connect to a remote VM running the software during its local installation to instantly access it.
   – made remotely accessible to the user. Due to SOP1 being a tablet with low resources, this option is chosen.

   Both options eventually result in the creation of a "launch" button on the user's desktop.

The second scenario, shown in the right-hand side of the same figure, describes the consequence of launching the software, i.e. the *deployment* of a VM running scilab.

1. The user 1 wants to use its freshly installed software. He presses the "scilab" button on its desktop and selects a scilab script he wants to run.
2. Given user's inputs, a Vishnu command "start job" is generated with the appropriate parameters.
3. Vishnu's meta-scheduler selects the most appropriate cloud to execute this new job and send a command requesting the instantiation of a suitable VM.
4. Said VM is instantiated and deployed on the location selected by SOPVP, the machine SOP2 of the user 2.
5. The scilab script submitted by the user 1 on his machine SOP1 is remotely executed in the new VM. Its result will be made available to user 1 when obtained.

Fig. 6 Scenario of migration

The third scenario is illustrated in Fig. 6. It consists in the shutdown of SOP2, leading to the migration of the VM to another cloud.

1. User 1 still has an execution of scilab on user2's device.
2. User 2 wants to stop her machine and activates the corresponding button, sending an event to the autonomic manager.
3. The stop of host sop2 is blocked until scilab migration
4. The autonomic manager checks whether VMs owned by another user are currently running on SOP2. If it is not the case, the machine is indeed stopped (11). Since user 1's VM is executed on SOP2, the autonomic manager starts the migration process. It disables machine SOP2. This state does not impact the execution of currently hosted VMs but forbid any new instantiation.
5. Different OpenNebula calls are handled by the autonomic manager:
   – each VM owned by another user, here the only considered VM, is put in the state reschedule. They will be treated by SOPVP during its next loop.
6. Since SOP1 can not support its execution and no other machine is available, the VM can not be executed within the same cloud. SOP VP notifies the autonomic manager for it to start the inter-cloud migration process. It sends an OpenNebula command to freeze and save the VM.
7. The autonomic manager sends a Vishnu command asking for the redeployment of a job and gets back the location selected by the meta-scheduler. It copies the created checkpoint to the selected location and creates a corresponding template there.
8. Vishnu sends an OpenNebula command to the selected cloud requesting the instantiation of a VM with the freshly created template.
9. The VM is instantiated within the targeted cloud on a location selected by SOPVP. An event is sent to the autonomic manager which will be treated in the (11) action.
10. User 2's VM running his scilab script is now remotely executed in the cloud provider.
11. The autonomic manager receives the event alerting it of the state of the new VM. It deletes the initial VM and, since there are no more VM running on SOP2, finally shutdowns the machine.

In this inter-cloud scenario a specific work is done to manage IP address of WM. we use a coherent mapping plan to avoid possible more complex solution using overlay address.

6.2 Schedulers

6.2.1 Experimental Platform

In this section, we will describe the experimental platform and methodology used to evaluate the allocation and reallocation strategies, and compare them against each other. We have done the experiments on a RECS testbed [12]. It comprises a single cloud deployed with OpenNebula over KVM, on 6 Intel i7-3615QE nodes (4 cores). Power consumption values are retrieved for each single processor with a Plogg watt-meter, thus giving us the ability to monitor each host (i.e: an i7) separately. In regards to OpenNebula, VM images are stored in a shared NFS storage, and each host has a capacity of 4 CPU, and 15.6 GB of RAM.

6.2.2 Methodology

Each experiment is done over the different scheduling algorithms. The first is the one that is default in OpenNebula, *mm_sched* with the striping heuristic. This setting is set by default because it is the one that will give the best QoS to the VMs, as it tries to allocate each VM to the host where there is the most resources.

The SOPVP scheduler is described in 4.2. For each algorithm, we collect several metrics. The power consumption as expected, but also the migrations number over time, and finally the relative duration of the VMs. The relative duration is computed as the actual duration from the moment the VM is actually running on the cloud, divided by the total duration, which is the duration from the spawn of the VM, until its deletion.

We randomly generate the workload over time on the cloud by randomly spawning VMs that are CPU stressed for their running duration. The random generation is done following a Poisson process, with a λ value tailored to generate an average load onto the cloud. The mean load of the cloud depends on the mean load generated by each VM. We will generate each VM to take between 0.1 and 2 CPUs, following a uniform distribution. Each VM will last for a duration between 300 and 500 seconds, uniformly generated, which is an average duration of 400 seconds. We will also generate the memory requirement for each VM to be between 1024 MB and 6624 MB. Each VM is designed to represent on average around 25% of a single host.

With those values in mind, we can compute the λ required for the Poisson process that will generate a specific average load.

Each VM is submitted to the cloud for the duration of the experiment, and the schedulers, that are on a periodic loop of 30 seconds, will allocate and reallocate them. Experiments between different schedulers are done using the same generated workload. Each set of settings is used on an experiment that will last 2 hours. That means that we've done a total of 17 different experiments, which represents more than 34 hours.

One should note here that a lot of variability inside a single run can occur. Indeed, there are small variations in the orders of actions that can take place when for example OpenNebula is responding to request, or even the variability in the time to boot a VM or a host, and yield to a different result.

6.2.3 Concurrent Migrations

We first start with an empirical experiment to calibrate the number of migrations we can do simultaneously on the testbed. To do so, we instantiate several VMs each with 2GB of memory, then we choose a fixed number of VM and migrate toward a random host. Obviously the network layout and the application inside the VM plays a big part in the result, but we need to keep in mind that it is only to define the number of hosts that we will unload in the algorithm. The RECS's hosts are all connected to a 1 Gbps switch. The last part of the data transfer, when the VM is stopped on the source host and the core of dirtied page is sent to the destination host [8], is done without bandwidth limitation. Theoretically, if all the memory of the VM is dirtied during the process of sending memory pages, migrating a single VM would take up to 18 seconds.

As we can see in Table 2, the average migration time of the virtual machines migrating concurrently from one host to another is increasing as the number of migration increases. This is possibly due to either the speed of the memory of the destination host, some contention in the network, and the time OpenNebula takes to proceed with the migrations.

Although the numbers themselves are specific to the platform and experimentation, it still tells us that at a certain point, concurrent migrations tend to be near a time that is unacceptable. This time is defined arbitrarily by the provider.

In our platform with this particular setup and VM size, we consider that 6 concurrent VM live migrations toward the same host is the maximum we aim to set.

Table 2 Time spend with concurrent live migrations

Nb. migrations	1	2	3	4	5	6
Avg. Time (s)	5.5	7.44	11.68	12.55	14.05	16.62

6.2.4 Implementing Energy Savings

There are several ways to handle the hosts state, that is when to power off and power on the physical machines to better save energy accordingly to what the VM consolidation algorithm is doing [19].

- FirstEmpty The first one is the easiest but not the most efficient, it works in a simple way, switching off hosts as soon as one host is empty, and power on hosts when the global load of the cloud is over a certain threshold.
- Pivot The second one works by keeping a pivot host. It first computes the projected number of host that is needed to run all the VMs, and tries to keep this number of hosts powered on plus the number of pivot hosts. This reveals the fact that the algorithm used for consolidation has to account

for the pivot host, otherwise it will wind up consolidating too much and having an empty host, as previously shown in [14]. Although in any case it allows some leeway in the reallocation process, which will diminish VM effective duration degradation.

– Variation The last host state management strategy we defined is based on the variation of the global load over time. It keeps the history of all the variations that occurred, and count how many of the loads are going up and down. If there is more down than up, it will try to power off a host, and vice versa.

We have tried out the different types of host management strategies and their effect on the energy consumption and average effective duration. Table 3 presents the results of both the energy consumption, where lower is better, and the effective duration, where a higher value is better. NoHM represents the baseline when we are not switching on and off the hosts. The PivotCeil is similar to the Pivot strategy with the number ceiled instead of rounded. The values are for a cloud with an average load of 50%, and the same SOPVP algorithm, which as seen before consolidates effectively. As we can see in Table 3, the most efficient strategy is the pivot, as it is the most tailored for this experiment in particular, and reacts the best to fast and slow load changes. We can save up to 23% of power while having a degradation of QoS of only 5%. We can also see that compared to the FirstEmpty approach, we gain with Pivot both in energy consumption (7%) and QoS (4%). The Variation strategy yields bad results, since for this kind of cloud load, it will switch on and off hosts, when there are already a sufficient number of host usable.

Table 3 Host Management Strategies

Strategy	noHM	FirstEmpty	Pivot	PivotCeil	Variation
Energy (kJ)	467	360	334	398	481
Duration (%)	84.58	77.24	80.73	88.86	87.40

6.2.5 Real Energy Consumption

In the last experiments we have implemented real powering on and off of the hosts which can be significant, as shown in [19]. We will conduct experiments three times with the host management strategy Pivot presented before.

Fig. 7 Energy consumption vs System Load, real energy consumption

Fig. 8 VM durations vs System Load, real energy consumption

As we can see in Figure 7, which represents the energy consumption for loads of 30%, 50% and 70%, the energy consumed is better using the SOPVP algorithm. We can also see that for higher loads, we can still save up 21% energy for a load of 50%, due to the reasons mentioned above, but also to a higher strain on the cloud, it is even more important to power on and off hosts.

Figure 8 presents the corresponding average effective duration of the virtual machines. We can see that we can maintain roughly the same amount of QoS as mm_sched, but that QoS is dropping as the load increases, which is to be expected. This drop is due to different reasons. For mm_sched, it's mainly due to the fact that on one hand we have the Pivot, which may be suited because it will take advantage always of the extra host if any, but on the other hand the global load on the powered on hosts which is higher induces the algorithm to fail to find suitable hosts for VMs, which leads to pending VMs, waiting to be allocated. That's also the reason why SOPVP is better at high loads, because there are more VMs, and it becomes increasing more likely that mm_sched will not find suitable host.

One should note here that even if an effective duration around 70-80% in average seems bad, it is mainly due to the fact that with VM duration around 400 seconds, a scheduling loop of 30s, 30s to power on an host, and between 30s and 1 minute for OpenNebula to acknowledge the host and add in to the host pool we can have a important impact of the waiting time that is due to the experimental platform, since it represents a high portion compared to the average duration. Such an effect would be greatly mitigated it we took VMs with higher mean durations.

SOPVP algorithm proposes a reallocation of VMs in a cloud, in order to save energy. We have compared it to the default algorithm provided in OpenNebula. We took into account several important factors to achieve consistent consolidation, such as the number of hosts to unload, directly related to the number of migrations we aim to make, but also the overhead induced by the powering on and off of the hosts, which takes time by itself in a real system. The algorithm is implemented in OpenNebula and has been experimented on a real platform.

7 Conclusion

This article proposes an extension of classical cloud model by considering user machine as part and parcel of its architecture rather than external requesters. Implications are twofold.
Firstly, available resources include said machines. User applications can thus be executed in three ways, ultimately reducing the load on the provider side: locally, on a remote datacenter, or on a remote user machine.
Secondly, autonomic management embraces user machines in addition to those of the datacenter. Users are consequently relieved of most administration tasks.

We defined a modular model and conformally implemented a platform realizing this new paradigm. It is composed by:

1. VISHNU to federate clouds. It dispatches and manages jobs.
2. OpenNebula to create each cloud and provide an API to perform actions on each cloud.
3. An autonomic manager implemented using FRAMESELF. It manages the platform by migrating VM between clouds, shutting down machines...
4. A market to simplify the access to new applications and manage billing.

To meet energy consumption and QoS concerns, a meta-scheduler and a scheduler have been proposed. The first has been integrated within VISHNU to select the most suitable cloud for each job. The scheduler, SOPVP, substitute OpenNebula's scheduler. Within a cloud, it selects the appropriate mapping between VMs and physical hosts.

Finally, this architecture and it usage have been modelled in the NEST simulation environment to design and size the capacity of the provider clouds to prevent overload.

Functionalities of the resulting platform are validated and demonstrated through various scenarios, three of which being described in this paper. The first depicts the purchase and installation of an application. The second illustrates its launch, from user request to VM instantiation. The third, more complex, begins with a user requesting its machine to be shut-down. Foreign VM are migrated to another cloud, due to a lack of resources on the current one. Once the required migrations realized, the scenario ends with the machine shut-down.
The internal scheduler managing resource usage is experimentally evaluated and compared with the native OpenNebula's scheduler. We took into account several important factors to achieve consistent consolidation, such as the number of hosts to unload, directly related to the number of migrations we aim to make, but also the overhead induced by the powering on and off of the hosts, which takes time by itself in a real system. Results show a significant reduction of energy consumption with almost no QoS degradation.

Extension of this work will be done to manage virtual machines that can integrate several applications. This will create new constraints in the scheduler and mode complicated rules for autonomic management.

Acknowledgements The work presented in this paper has been funded by the ANR in the context of the project SOP, ANR-11-INFR-001.

References

1. M. Bencivenni, D. Michelotto, R. Alfieri, R. Brunetti, A. Ceccanti, D. Cesini, A. Costantini, E. Fattibene, L. Gaido, G. Misurelli, E. Ronchieri, D. Salomoni, P. Veronesi, V. Venturi, M. Vistoli. Accessing Grid and Cloud Services Through a Scientific Web Portal In *Journal of Grid Computing, Springer Netherlands*, Vol. 13, no. 13, pp 159-175, 2015.
2. Vishnu Distributed Resource Management Middleware. http://sysfera.github.io/Vishnu.html
3. QoS Design Network Simulation Tool (NEST). http://www.qosdesign.com/index.php?menu=produits&langue=fr
4. R. Moreno-Vozmediano, R. S. Montero, I. M. Llorente. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer*, vol. 45, no. 12, pp. 65-72, IEEE, 2012.
5. Pieter Hintjens. ZeroMQ library: *http://zeromq.org/*, November 2014.
6. A. Yoo, M. Jette, M. Grondona. Job Scheduling Strategies for Parallel Processing, volume 2862 of *Lecture Notes in Computer Science*, pp 44-60, Springer-Verlag, 2003.
7. IBM. Tivoli workload scheduler loadleveler. http://www-03.ibm.com/software/products/fr/tivoliworkloadschedulerloadleveler
8. Kvm live migration. *http://www.linux-kvm.org/page/Migration*, June 2014.
9. OpenNebula: Flexible enterprise cloud made simple. *http://opennebula.org/*, June 2014.
10. OpenNebula: Match making scheduler. *http://archives.opennebula.org/documentation:rel4.4:schg*, June 2014.
11. Plogg smart plugs. *http://www.plogginternational.com/*, June 2014.
12. The RECS testbed. *http://recs.irit.fr/doku.php?id=recs:features*, June 2014.
13. Anton Beloglazov, Jemal Abawajy, Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener. Comput. Syst.*, Vol. 28, no. 5, pp. 755-768, May 2012.
14. Damien Borgetto, Michael Maurer, Georges Da Costa, Ivona Brandic, Jean-Marc Pierson. Energy-efficient and SLA-Aware Management of IaaS Clouds. In *ACM/IEEE International Conference on Energy-Efficient Computing and Networking (e-Energy).* ACM DL, 2012.
15. M. Caballer, I. Blanquer, G. Molto, C. de Alfonso. Dynamic Management of Virtual Infrastructures In *Journal of Grid Computing, Springer Netherlands*, Vol. 13, no. 1, pp 53-70, 2015.
16. C. Ghribi, M. Hadji, D. Zeghlache. Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms. In *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 671-678, May 2013.
17. Michael R. Hines, Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *13th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pp. 51-60, 2009.
18. M. Ben Alaya, T. Monteil. FRAMESELF: An ontology-based framework for the self-management of M2M systems. *Concurrency and Computation: Practice and Experience*, Wiley, 2013.
19. Laurent Lefèvre, Anne-Cécile Orgerie. Designing and Evaluating an Energy Efficient Cloud. *Journal of Supercomputing*, Vol. 51, no. 3, pp. 352-373, March 2010.
20. W. Leinberger, G. Karypis, V. Kumar. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *International Conference on Parallel Processing.* , pp. 404-412, 1999.
21. S. Takahashi, A. Takefusa, M. Shigeno, H. Nakada, T. Kudoh, A. Yoshise. Virtual machine packing algorithms for lower power consumption. In *IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 161-168, Dec 2012.
22. G. Bacso, A. Visegradi, A. Kertesz, Z. Nemeth, Zsolt. On Efficiency of Multi-job Grid Allocation Based on Statistical Trace Data In *Journal of Grid Computing, Springer Netherlands*, Vol. 12, no. 1, pp 169-186, 2014.
23. IBM: An Architectural Blueprint for Autonomic Computing,. IBM White Paper 4th Ed., June 2006.

24. Zhen Xiao, Weijia Song, Qi Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 24, no. 6, pp. 1107-1117, June 2013.
25. Jyun-Shiung Yang, Pangfeng Liu, Jan-Jan Wu. Workload characteristics-aware virtual machine consolidation algorithms. In *IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 42-49, Dec 2012.
26. Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, Davide Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia. A view of cloud computing, *Commun. ACM*, Vol. 53, no. 4, pp. 50-58, April 2010.
27. Rimal Bhaskar Prasad, Choi Eunmi, Lumb Ian. A taxonomy and survey of cloud computing systems, *Fifth International Joint Conference INC, IMS and IDC (NCM'09)* , pp. 44-51, 2009.
28. Mell Peter, Grance Timothy. The NIST definition of cloud computing (draft), *NIST special publication 2011, vol. 800(145)*, 2011.
29. Soundararajan Vijayaraghavan, Govil Kinshuk. Challenges in Building Scalable Virtualized Datacenter Management, *SIGOPS Oper. Syst. Rev.*, Vol. 44, no. 4, pp 95-102, December 2010.
30. H. Ben Cheikh, J. Doncel, O. Brun, B.J. Prabhu. Predicting Response times of applications in virtualized Environements, *3rd IEEE Symposium on Network Cloud Computing and Applications (IEEE NCCA2014), Rome, Italy, February 2014.*