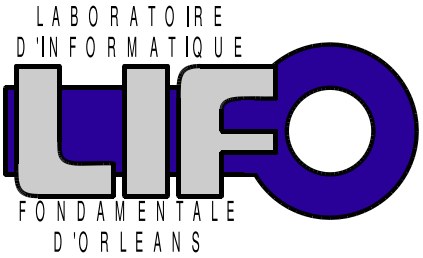

4 rue Léonard de Vinci
BP 6759
F-45067 Orléans Cedex 2
FRANCE
http://www.univ-orleans.fr/lifo

Rapport de Recherche

A Trust Aware Distributed and Collaborative Scheduler for Virtual Machines in Cloud

Jonathan Rouzaud-Cornabas
LIFO, ENSI de Bourges

Rapport n^o **RR-2011-09**

A Trust Aware Distributed and Collaborative Scheduler for Virtual Machines in Cloud

Jonathan ROUZAUD-CORNABAS
Laboratoire d'Informatique Fondamentale d'Orléans
Ensi de Bourges – Université d'Orléans
88 bd Lahitolle, 18020 Bourges cedex, France
jonathan.rouzaud-cornabas@univ-orleans.fr

May 11, 2011

Abstract

With the number of services using virtualization and clouds growing faster and faster, it is common to mutualize thousands of virtual machines within one distributed system. Consequently, the virtualized services, softwares, hardwares and infrastructures share the same physical resources, thus the performance of one depends of the resources usage of others. Furthermore, each user and his VMs have different objectives in term of quality of trust and protection. Thus, the placement and reconfiguration processes need to take them into account. Finally, a new trend is to use clouds to build HPC systems but the deployment of such architecture is not straightforward.

We propose a solution for VM placement and reconfiguration based on the observation of the resources quota and the dynamic usage that leads to better balancing of resources. Moreover, our solution also takes into account user's objectives that express Quality of Trust and Protection and ease the deployment of HPC architectures on top of clouds. Furthermore, as it is not possible to have a single scheduler for the whole cloud and to avoid a single point of failure, our scheduler uses distributed and collaborative scheduling agents. Finally, we present a scenario simulating a geographical distributed clouds and multiple VM usage. This article is an extended version of [22]. It includes larger scale evaluations and the scheduler supports user's objectives in addition of the resource-centric algorithms.

1 Introduction

Nowadays, server farms are popular for running a large range of services from web hosting to enterprise systems or even HPC clusters. The common way to deal with those growing server farms is to mutualize services, softwares, hardwares and infrastructure using virtualization technologies. These mutualized and virtualized infrastructure are named clouds.

The concept of cloud computing goes back to 1961 with the introduction of Utility as a service by John McCarthy. The goal was to bill computing power as water or electricity. The computing power would have been provided by a large infrastructure such as a shared grid. In the past, such infrastructure was impossible due to high costs but nowadays, with cheap and efficient hardware, it has become possible. However, when multiple virtual machines share the same physical resources, the performance of each VM and its embedded application depends on the resources usage of other VM running on the physical host. So, the management of VM becomes critical. Currently, most of clouds (and grids) schedulers are solely based on quota negotiations and do not take into account real resources usage.

Furthermore, each user within the cloud uses it to run a different set of applications and each applications are used for different purpose. Accordingly, each VM has different quality objectives. Thus, to cope with this issue, the cloud must take into account objectives that are given for each VM by their owner and enforces them.

2 Security and Trust within Clouds

Security is one of the top concern in clouds [20]. Indeed, the security perimeter becomes blurred in cloud e.g. the delimitation between intranet and internet. One of the critical issues in cloud security is the geographical localization of computing and data processes. Indeed, moving data from a country to another one can lead to multiple issues [12, 18]. It is very difficult to know which laws and judicial jurisdiction apply in case of lawsuits between the Cloud Service User and the Cloud Service Provider. Thus, the geographical location becomes critical in clouds [23]. Moreover, moving a software in the cloud can be illegal if the cloud is not certified by a special standard e.g. PCIDSS for the card payment software. The first step to bring security in cloud is to have an user's objectives (policy) aware scheduler [24]. Indeed, the scheduler is the first step to increase security as it places and reconfigures the task i.e. the virtual machines, within the infrastructure.

Another concern within clouds is the trust issue. Cloud trust can be defined by "trust without touch" [17]. Indeed, the data are no more within the company or the user computer but on an outsourced infrastructure. Accordingly, the data is on hardware that is outside the scope of the entity that owns it. There are two issues related to trust [15]: the lack of transparency of clouds' infrastructure and unclear security assurances. One example of lack of transparency is that an user does not know where is his data i.e. in which datacenter and what is the level of security of each datacenters. Thus, audit traces are needed to help the user to check if his objectives are respected. But these traces need to not expose too much informations that could lead to information leakages [21]. Furthermore, the level of security of datacenters can not be given by the operators of the datacenter themselves but needs to be certified by a third party. Many certification processes already exist for this purpose such as Tier grade datacenter or ISO-27001 standard. Moreover, the level of trust required by users is not always the same. For example, a 99.9% SLA is enough for most companies and users but it will not be enough for an emergency service like 911. Finally, SLA alone is not enough, an user must be able to express other objectives. For example, to reduce the risk of inter-VM attacks [21] due to the lack of security in the virtualization layer, an user can express objectives to have dedicated physical machines within the cloud.

3 Schedulers and Virtual Machine

3.1 Scheduling on distributed systems

The scheduling of jobs on a distributed system such as a grid or a cloud is a NP-Complete problem [8]. Moreover, the result of the scheduling process is not optimal too. It means that the schedulers are not searching for the optimal solution but a one that is good "enough" [3].

The scheduling algorithms can be divided into two main categories: static and dynamic. Static algorithms are based on the prediction of the process behavior whereas dynamic algorithms probe resources before taking the scheduling decision. Dynamic algorithms are useful when it is not possible to predict the behavior of a task [8] and are the best algorithm to maximize the resource usage.

The architecture of schedulers can be divided into two main categories. The centralized approach uses a single scheduler that has access to all the information and takes decisions alone. The decentralized approach uses multiple scheduler agents (up to one per host) to take the decision. One kind

of decentralized schedulers are P2P schedulers [10]. Within the peer-to-peer model, the cloud is seen as a set of nodes (i.e. peers) that make available for each other a part (or the totality) of their resources. In contrast to the client-server model, with the P2P model, each computer in the distributed system is supplying and using the resources. The main advantage of the P2P is that it does not contain a central coordination point and thus, avoid a single point of failure.

3.2 Distributed Scheduler

The main approaches to distribute the load between schedulers [6] are: work sharing and work stealing. They are nearly opposite in their design. Work sharing is inherently centralized whereas work stealing is distributed. Work stealing has a better scalability and fault tolerance through the use of a decentralized approach [19]. The work stealing method allows idle consumers to search among the other consumers to find additional work. The tasks of finding and moving tasks to idle consumers is done on the idle consumers them self. Through, the overhead is minimized to idle consumers with free resources. The authors of [7] have done scalability experimentations that show that the algorithm scales well and is, at least, as efficient as work sharing.

3.3 Virtual Machine (VM)

The live migration process [4] allows a virtual machine to move from a host to another one without being stopped. From a virtualization point of view, a cloud is seen as a shared and distributed environment where concurrent users run VMs on. Those VM have a heterogeneous behavior e.g. a website with peak traffic or a graphical interface. Accordingly, the resource usage can change at any given time, so taking into account the dynamic resources is essential. There is two approaches for detecting resources bottleneck: blackbox and greybox. Blackbox is based on the resources usage of the VM and is operating system agnostic. Greybox is based on instrumentation of the OS and applications and is not OS agnostic.

3.4 VM Schedulers

A VM scheduler places VMs on physical machines within the cloud. Furthermore, they can reconfigure the VM placement through the use of live migration to optimize global goals on the cloud. These goals range from energy reduction to performance maximization.

Entropy [13] is a virtual machine scheduler dedicated to consolidate a cluster. Entropy's goal is to reduce the number of physical machines that are running VMs to reduce the energy consumption. Moreover, it takes into account the cost of VM migrations to avoid unnecessary ones. This is done by taking into account the amount of memory used by the VM. Indeed, the live migration time duration for a VM is linked to its memory usage. But Entropy is dedicated toward one goal, energy consumption, and does not take into account heterogeneous architecture. Furthermore, the placement is based on resources' quotas allocated to VM, thus it does not take into account real resources usage and user's objectives.

Snooze [9] is another virtual machine scheduler dedicated toward reducing the energy consumption by minimizing the number of physical machines in the cloud. It uses a heuristic approach to find a placement solution. But, contrary to Entropy, their solution is not centralized. Snooze is scalable and fault-tolerance. But as Entropy, it is dedicated to one goal (energy consumption) and can not take into account other ones.

In [27], the authors present a two-level cloud scheduler. First, the VM provisioning process places applications into a set of VMs. Then, the VMs are placed on physical machines. To solve the provisioning and placement problems, they use a constraint solver. Their placement time is between 800ms and 1,000ms. Moreover, the user's objectives are limited to one metric such as average response time of requests. Thus, they are not able to express high level objectives.

In [1], the authors present a software dedicated to the management of VM migration for the consolidation of a cloud. The scheduler's goal is to choose which VM to migrate and when. As explain in [4], a local migration uses only few milliseconds whereas it uses 1-2 sec on a WAN [26]. This migration time can be reduced by the use of an infiniband network as explain in [14]. Other solutions for VM migration [29, 2] exist but they use threshold whereas the one proposed in [1] studies the VM usage history thus reducing the number of migrations.

4 Motivation

There is some automatic solutions that allow to share the load of multiple VMs on a heterogeneous and distributed system but they mainly are dedicated toward one goal, reducing the energy consumption.

As VM behaviors can not be predicted due to their complex behaviors and non-deterministic events, dynamic approach is the best choice. In clouds,

"black-box" monitoring is needed because the VMs are heterogeneous and instrumenting them is too time-consuming. Moreover, with VMs provided by users, instrumentations required inside VMs to trust all the users within the cloud and that is impossible. Furthermore, clouds require the placement and reconfiguration processes to be operating systems and applications agnostic to cope with heterogeneity. In addition, the placement must remain "good" enough for the maximum period of time to reduce the cost due to VM migration. Finally, building dynamic scheduler on a cloud brings a scalability challenge as one scheduler can not allocate VMs for the whole cloud. But, distributed dynamic scheduler can achieve such goal.

Moreover, as we explain, all the users on a cloud do not have the quality objectives. Indeed, some want to use it to run critical programs whereas other want cheap computing power. Thus, we need a way to let the users express their quality objectives. In this work, we focus on goals that enable the user to express their objectives in term of quality of trust and protection but also ease the use of cloud to create HPC cluster or grid.

To reach the goal of a decentralized virtual machine scheduler for clouds that takes into account user's objectives, we introduce our architecture that enables:

1. The monitoring of the VM and the hosts;
2. The placement of the VM on the best fitting host based on dynamic resources-centric algorithm and user's objectives;
3. The reconfiguration of VM if the load on a host is increasing, a special resource [1] is needed or a maintenance operation is taken place. The reconfiguration must keep the enforcing of user's objectives.
4. The scheduler must not contain a single point of failure and must scale with the cloud infrastructure. This can be done through the use of a distributed P2P architecture where the scheduler agents cooperate to share the load of the placement processes based on a work stealing algorithm;

Furthermore, in [16], the authors state that traditional distributed and dynamic scheduling methods have the following issues: they do not assume the heterogeneity of used hardware; processes, i.e. VMs in our case, are assigned onto nodes based on nodes free resources only; many of them consider just the processor utilization as the only resource. Thus, our dynamic scheduling

[1] For example, a GPU.

algorithm takes into account the heterogeneity of the hardware and of the VMs and also free, static and quota resources for the processor and memory utilization.

5 Scheduler Architecture

A cloud is composed of multiple nodes that can be heterogeneous. The vast majority of these nodes are computing nodes that can run multiple Virtual Machines (VM) through a hypervisor. In our architecture, this kind of nodes are named *Hypervisor Nodes* or HN. Another kind of nodes are the ones that place newly submitted virtual machines. These nodes are named *Scheduler Nodes* or SN. In our evaluations, we run one SN per datacenter i.e. one per geographic location. But, these two kind of nodes can be combined. In this case, they do both: place and host virtual machines.

In our architecture, each node provides a set of services. Each of these services is a component that provides a part of our scheduler architecture. They are described in the section 5.1. All the nodes provide a set of services and use another set of services. Thus, they are both consumers and suppliers of resources i.e. the services. Moreover, to avoid the complex task of configuration within our architecture, all the services are found through dynamic discovery. Thus, the whole architecture is self-configured. Furthermore, our architecture copes with the dynamicity of cloud as it easily discovers new and deleted resources. To fit those needs, the peer-to-peer architecture is the best approach. Accordingly, all the nodes of our architecture are connected through a peer-to-peer system.

One service (MScore) is reachable through multicast. It brings an efficient scaling service as it allows to query a set of nodes with one request and receiving all the responses in parallel. As we present later, it efficiently reduces the placement and reconfiguration time.

Finally, to increase the decentralized capabilities of our architecture, we uses a work stealing algorithm as described in the section 5.2. It allows to share the task of placing virtual machines on the cloud.

5.1 Services

As we previously say, all the components needed for our architecture are supplied by services. We describe here each of these services and what they do.

- **Queue**: It is the entrypoint of the cloud i.e. it is there that to-place

VMs are submitted. The service is reachable through the cloud but also by the users outside the cloud. It is important that this service can be reach through the cloud as it allows a VM to request new VMs. It helps to provide vertical elasticity i.e. duplicating virtual machines to share the load.

- **Score**: It computes the score of the HN where it runs and returns it back.
- **MScore**: It is the same service as the Score one but it is reachable through a multicast address.
- **LaunchVM**: It receives a request to start a VM with all the related parameters: VM quotas (quantity of memory, number of CPU cores, etc) and the user's objectives. Then it checks if it can start the VM and runs it. If it can not start the VM, it migrates one (or more) VM(s) to be able to start it.
- **DHT**: It uses the Chord [25] to store VMs' metadata into a distributed hashtable. It allows any services to know for each VM its quotas, objectives and where it is running.
- **Monitor**: This service receives monitoring events from all the other services. It provides traceability for the scheduling architecture. For example, it allows a user to known when and where each of his VMs have been place. Another use case of this service is to allow to verify that the whole process of placing a VM does not have encounter failure.
- **Migrate**: It receives a VM and starts the live migration process. The VM is migrated from the HN that calls the Migrate service to the HN that hosts the service.
- **WorkStealing**: It allows the work-stealing process by exposing a shared queue of to-place VM.

The *Hypervisor Agent* or **HAgent** runs on all HNs. It is composed of the following services: Score, MScore, LaunchVM, DHT and Migrate. Moreover, it contains an internal service, SystemMonitor, that monitors the resources usage and detects bottleneck. It allows to migrate VMs when the load on a HN is to high.

The *Scheduler Agent* (SAgent) runs on all SNs. It is composed of the following services: Queue, DHT, Monitor and WorkStealing. Furthermore,

it contains an internal service, QueueMonitor, that takes to-place VMs from the local Queue service and launches the placement process for them. When QueueMonitor can not retrieve to-place VM from the local Queue i.e. it is empty, QueueMonitor steals a bunch of to-place VMs from another SAgent.

To avoid the use of dedicated nodes for the scheduling process, the four services (Queue, Monitor, QueueMonitor and WorkStealing) can be added to the HAgent.

5.2 Decentralized Scheduler

Our work stealing algorithm implementation is straightforward. First, each SAgent has two queues:

- a **local queue:** for the to-place VMs that the local SAgent is placing itself i.e. a list of task reserved for the local scheduler.
- a **shared queue:** for the to-place VMs that can be placed either by the local SAgent or by other ones through the use of work stealing.

To fit it in our decentralized peer-to-peer system, the second queue is made available as a service for other SAgents i.e. the WorkStealing service.

The workstealing process works as follow, the local agent reserves a set of to-place VMs for itself (by removing them from the shared queue and adding them to the local queue). When the local agent has placed all the virtual machine in the local queue, it preempts another set. If there is no more tasks in the shared queue, it randomly chooses a victim SAgent and tries to take a set from it. By having the two queue (local, shared), we have a better scalability with a limited latency due to lock. Moreover, we respect our goal of no point of failure by using a distributed algorithm.

6 Election Algorithms

The purpose of the election algorithms is to elect a HN or VM. In both placement and reconfiguration process, a HN needs to be elected to host the VM. This HN needs to be selected based on two things: first, it respects the user's objectives and second, it will provide good performances for the VM. The election of a VM is needed when we need to move a VM from a HN to another but we do not know which VM to move as it will be explain in the section 7 and 8.

Each VM are defined by a set of user's objectives and two types of resources quota for processor and memory. The first type of quota, soft, is

the quantity of resources that is dedicated for the VM. The second type of quota, hard, is the quantity of resources that can be used by the VM but is not dedicated to it. Thus, based on resources consumptions of other VMs, the resources can be used or not.

Each HN are defined by a set of goals they can reach. For example, do they have the ability to host a HPC VM. They also include their geographical location and their resources (processor, memory, etc).

6.1 User's Objectives

The user's objectives is a set of goals that an user can specify for his VMs. We have implemented a set of objectives that allow an user to define quality of trust and protection. The purpose of these objectives is to able the user to choose the level of trust and security he want for his VMs on the cloud. Of course, with the increase of trust and security, the price of running the VM increases. Also the objectives eases the provisioning of virtual HPC cluster or grid on top of a cloud. The objectives are also used to describe the capabilities of HNs e.g. if PCIDSS is defined for a HN, it means that the HN is PCIDSS certified. We indifferently use HN capabilities and HN goals to refer to the capacity of a HN to fulfill an objective.

We have defined 5 objectives related to trust:

- **tier**: The *TIA-942:Data Center Standards Overview* describes the requirements of a datacenter architecture. Based on this standard, the datacenters are graded from 1 to 4 where 1 is the simplest datacenter and 4 the strongest one. Based on the tier grade, the datacenter can be more or less trusted. Indeed, with the increase of tier grade, the datacenter includes more and more fault tolerance equipments and an increase availability and security. Of course, the highest tier grade datacenter are more expensive than the lowest one. Accordingly, an user can choose the highest or lowest tier grade based on his needs for each of his VM. This goal is defined as an integer between 1 and 4.

- **PCIDSS**: It stands for Payment Card Industry Data Security Standard and consists of 12 requirements for all businesses that want to process card payment transactions. Thus, this objective is needed if an entity want to create a card payment system on top of the cloud. Of course, it only provides the infrastructure to build it and the top software layers need to fulfill PCIDSS too but it enables the ability to build PCIDSS approves infrastructure on top of a cloud. This goal is defined as a boolean.

- **ISO-27001**: It is a standard for Information Security Management System (ISO/IEC 27001:2005 - Information technology – Security techniques – Information security management systems – Requirements). The purpose of ISO-27001 is to certify the quality of protection of an infrastructure. For an user, running a VM on an ISO-27001 infrastructure can be required for his entirely infrastructures or just a part of it. This goal is defined as a boolean.

- **EAL**: The Evaluation Assurance Level of a system is a grade given following the completion of a Common Criteria evaluation. With the highest grade, the system provides higher security features. It ranks the level of security brought by the system. The requirements that must be reach for each grade, involve different measures from penetration testing to design analysis. Based on his needs, the user can specify the level of security for his VM. Of course, as high grade systems are more expensive, running VMs on them is more expensive too. This goal is defined by an integer between 1 and 7.

- **SLA**: A Service Level Agreement is a part of the service contract between an user and a service provider. In the case of the cloud, it is a contract between the Cloud Service Provider (CSP) and the Cloud Service User (CSU) that defines the quality of service guaranteed on the cloud (or a part of the cloud). Currently, most of SLA on clouds contain just the expected uptime i.e. the mean time between failures but SLA can be more complex. For example, the project MyCloud supported by a grant from the French National Research Agency has proposed SLA for clouds [28]. Other projects have done the same like the European project SLA@SOI [5]. This goal is defined by a float that ranges between 0 and 100.

We have 4 goals for protection. The first three are related to geographical location and the fourth increases the isolation of an user on the cloud. The location is very important in term of protection as it defined which judicial jurisdiction will applied for the data and its processing within the cloud. Indeed, the laws change from country to country and thus it can forbid that data and its processing are outsourced to another country. To tackle this issue, an user can specify, by using the location objectives, where he want to run his VMs. But, enforcing a VM location decreases the consolidation of the cloud as it reduces the placement to a smaller set of HN. Accordingly, more the user specifies a specific location e.g. a datacenter, more it will reduce the consolidation. Geographical location can also help to ease the creation

of fault tolerance and resilient systems that are built on top of a cloud as an user can specify to span an architecture into different geographical locations. The fourth goal allows an user to ask to be alone on a HN. Thus, when his VMs run on a HN, he knows he is the only user on it. It helps to reduce the risk of inter-VM attacks that pass through the virtualization layer as explain in [21]. Of course, it dramatically reduces the consolidation of the cloud as some of the HN are dedicated to one user.

Finally, we have defined four objectives for HPC. The goal is to ease the deployment of virtual HPC cluster or grid on top of a cloud.

- **HPC**: It allows to specify that the hardware must be HPC grade material. For example, classic HN could included no-ECC memory but all the HPC HN will included ECC memory. Moreover, on HPC HN, the processor allocated to a VM is not a part of a virtual CPU but a dedicated physical processor. Thus, the user accesses more directly to the underlying hardware, thus it limits the overhead that is due to virtualization layer. This goal is defined as a boolean.

- **HPN**: It allows to specify that the hardware must include high throughput and low latency network card such as Infiniband. The Infiniband cards are dedicated to one VM that accesses it directly without any virtualization layer. It eases the deployment of a HPC architecture with RDMA capability [2]. This goal is defined as a boolean.

- **GPU**: It allows to specify that the hardware must include a dedicated GPU allocated to the VM. As more and more HPC architectures include GPU, a cloud that ease the deployment of such architecture needs to be able to provide such materials on-demand. This goal is defined as a boolean.

- **nearVM**: With HPC architecture, high throughput and low latency are discriminant criteria but the geographical distance between two VMs biases the quality of the architecture. Thus, with this goal, an user can required that two (or more) of his VMs are running the more closely (in term of geographical location) that it is possible. For example, all the VMs could run on the same HN or different HN within the same rack. Of course, as many goals, it reduces the overall consolidation of the cloud and thus must be charged back to the user. This goal is defined as a boolean.

[2]Remote Direct Memory Access (RDMA) is a direct memory access from one computer to another without passing through the operating system.

Using these goals, an user can specify its requirements and build his custom architecture on top of the cloud.

6.2 HN Score

Using the static, free and used resources of a host and the running VM combined with the soft and hard resources quotas, we are able to compute a score that is more relevant than a classical weight given to each host based on their static resources (and other static metrics).

In order to balance the load between the HNs of our architecture, a score is computed for each node. The computation of the score needs to be quick, does not require too much resources and does not need a central controller or repository. But at the same time, the score needs to closely reflect the amount of resources used by each VM. To reach those goals, we introduced our score algorithm. Furthermore, it can be extended on-demand to take into account other resources than CPU and memory usage like network or hard drive bandwidth. This score is divided into 2 parts. A **static score** that takes into account static resources e.g. the amount of CPU core on a node and the resources (soft and hard) quota reserved for virtual machines. A **dynamic score** that is based on dynamic resources e.g. the amount of free memory.

As shown in the listing 1, the algorithm to compute the static score takes two arguments: a structure, *host*, that contains the static resources of the node and a list of virtual machines, *vmlist*, that contains VM resources' quotas. For each virtual machine in the list, the RAM and CPU soft quotas are added and multiplied by a static parameter α [3], then the result for each virtual machine is added into the variable $static\_vm\_soft$. The same thing is done with the hard quota and the static parameter β into the variable *static_vm_hard*. Secondary, the *static_host_score* is computed by dividing the processor speed $host(CPU)_{speed\_in\_Mhz}$ by a static parameter γ then it is multiplied by the number of core $host(CPU)_{number\_of\_core}$. The final static score *static_score* for the given *host* with the list of virtual machine *vmlist* is computed by summing the amount of memory to the result of last computation.

Listing 1: "Score computation algorithm"

```
compute_score(host)
  static_vm_soft = 0
  static_vm_hard = 0
  static_score = 0
```

[3] All the static parameters are set through experimentations (see section 9).

```
  for each VM xvm in host.vm_list
  do
    static_vm_soft += ( soft_cpu_quota(xvm) + soft_ram_quota(xvm) ) x α
    static_vm_hard += ( hard_cpu_quota(xvm) + hard_ram_quota(xvm) ) x β
  done

  static_host_score = ram_size(host)
      + size(host.cpu_list) x (cpu_speed(host) / γ)

  if ! ( static_vm_soft > static_host_score )
  then
    static_score = static_host_score - static_vm_soft

  if ! ( static_vm_hard > static_host_score )
  then
    static_score = ( δ x static_score )
        + ( static_host_score - static_vm_hard )

  return static_score
```

The computation shown in Listing 2 is the dynamic part of the score. It takes into account the free resources of the HN and the resources used by the running VMs. Our approach is based on a global scheduler on all cores of each HN but our score can be extended to take into account the need of a dedicated core (or a part of it). For each CPU cores, it gets the amount of free CPU and sums them. Then it multiplies this result by the static variable (γ) and sums the result to the amount (in Kb) of free RAM. For each virtual machine, it retrieves the amount of CPU and memory used. Then it multiplies the amount of CPU used by all VMs by a static value (β) and then sums it with the amount of free RAM. Finally, the dynamic score is computed by dividing the amount of free resources by the number of used resources by the VM.

Listing 2: "Score computation algorithm"

```
dyn_score(host):
  free_cpu_total = 0; used_cpu_total = 0; used_ram_total = 0
  for each CPU core xcpu in host.cpu_list:
    free_cpu_total += free_cpu(xcpu)
  for each VM xvm in host.vm_list:
    used_cpu_total += used_cpu(xvm)
    used_ram_total += used_ram(xvm)
  dyn_score = ( free_cpu_total x γ + free_ram(host) )
              / ( used_cpu_total x β + used_ram_total )
  return dyn_score
```

To compute the resource-centric score $score(host)$, the static part of the score is multiplied by a static value (κ) and then adds to the dynamic score that gives the score of a given node. Our approach permits a better place-

ment and helps to limit the needs of migration by taking into account resources that can be potentially used by the VM (the soft and hard quota).

But this resource-centric score is not enough, we need to take into account the user's objectives. This process is divided into two parts: first, is the HN fulfill the user's objectives (else the score must be 0) and second, what is the ratio between the capabilities of the HN and the VM's requirements. Indeed, we want to limit the number of VMs that are place on a HN that overfits its requirements. We prefer to keep those HNs for VMs that really need the full capabilities of the HNs. For example, if a VM does not required a HPC HN but the one with the highest resource-centric score is a HPC HN, we want to see if a good enough HN is available without HPC capabilities to place the VM. This requirement is due to the cost of reconfiguration. Indeed, when another VM will need the HPC HN, we will need to migrate the first VM to a new HN. Thus, it is simpler and costs less in term of migration to directly place the VM on a HN that does not overfit too much the VM's objectives.

The function $fitVM(host, VM)$ checks if a HN, *host*, provides all the capabilities required by a virtual machine, VM. If the HN fulfills the goals then the function returns 1 else 0. Both variables VM and *host* provide a set of functions to browse their goals and capabilities. The $def...()$ functions allow to know if a given goal or capability is defined. For example, $defQualTier()$ permits to know if one of the VM's objectives is to have a minimum datacenter tier grade. Another set of functions $get...()$ permits to retrieve the value of an objective. For example, $getQualTier()$, retrieves the tier grade asked (from 1 to 4). For all the objectives defined as boolean, if the objective is defined for the VM e.g. $VM.defQualPCIDSS() = True$, then it must be defined for the HN e.g. $host.defQualPCIDSS() = True$, and their values must be the same i.e. $host.getQualPCIDSS() = VM.getQualPCIDSS()$. For the objectives define as integer or float, the objective must be defined for the VM and for the HN as for boolean objectives. Moreover, the value of the objectives for the HN must be higher or equal than the one defined for the VM i.e. $VM.getQualTier() \leq host.getQualTier()$.

The function $overfitVM(host, VM)$ returns a metric that represents the fitting of a VM on a HN. As we explain earlier, the purpose of this function is to avoid to place a VM on a HN with too many capabilities that VM will not used. The function is presented in the equation 1. First, it does an intersection between the set of goals of the VM and the goals of the HN and computes the size of the resulting set. Then it divides this result by the size of the set of HN's capabilities. Finally, it subtracts the resulting number to 2 and returns the result of the computation. The return is a float between

1 and 2 where 2 indicates that the VM fits perfectly on the HN and 1 that the VM does not use any capabilities given by the HN.

$$overfitVM(host, VM) = 2 - \frac{|VM.goals\_set \setminus host.goals\_set|}{|host.goals\_set|} \quad (1)$$

Finally, we use the three functions $score(host)$, $fitVM(host, VM)$ and $overfitVM(host, VM)$ to compute the HN score. The equation 2 describes this computation. It multiplies the result of the three functions.

$$hn\_score = fitVM(host, VM) \times score(host) \times overfitVM(host, VM) \quad (2)$$

6.3 VM Score

The VM score is used to elect a VM when a migration must happen but it is not targeted toward a specific VM. First, we want to avoid to migrate a VM that fits perfectly on a HN in term of objectives as it will be more difficult to find another HN for it. Second, as the migration cost increases with the amount of memory used by the virtual machine, we need to take it into account. Third, we want to avoid circular migrations e.g. when a VM A is migrated to make space for a VM B that has been migrated to make space for A. To avoid such issues, we have a list of VM, $prevVM$, that contains all the VMs that have been migrated to perform a reconfiguration. Thus, we avoid to move a VM multiple times in the same reconfiguration process.

As for the HN score, the VM score is divided into two parts: resource-centric and user's objectives. For the user's objectives part, we reuse the function $overfitVM(host, VM)$ presented in the equation 1 to compute the fitting of the VM on the current HN. Then, we use the algorithm presented in the Listing 3 to compute VM score. The first thing is to return 0 if the VM has already been move by this reconfiguration process. Then we compute the $overfitVM()$ score and the resource-centric score, dyn_score. $VM.free\_cpu$ returns the number of megahertz between the soft quota of processor and the number of megahertz currently used by the VM. $VM.free\_ram$ does the same for the memory. This value is in megaoctet. $VM.used\_cpu$ and $VM.used\_ram$ returns respectively the number of megahertz and the number of megaoctet used by the VM. γ and β are set to a value smaller than 1 (0.05 in our evaluation) because as we explain before it is the amount of memory used that impacts the time of migrating a VM. Finally, the dyn_score is multiply by the overfitting score.

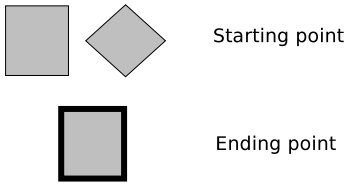

Figure 1: Legend for the flowcharts

Listing 3: "VM Score computation algorithm"

```
vm_score(host,VM):
  if VM in prevVM:
    return 0

  ovm = overfitVM(host,VM)
  dyn_score = ( VM.free_cpu_total x γ + VM.free_ram )
                / ( VM.used_cpu x β + VM.used_ram )

  return ovm * dyn_score
```

7 Placement

Our architecture takes into account user's objectives and a dynamic-resources centric algorithm to place the VMs. In this section, we describe how the different steps of the placement process are working. All the flowcharts, in this section and in the section 8, use the same legend presented in the figure 1.

As described in the figure 2, all the new to-place VMs are sent to the Queue service. Then the Queue service adds them into the shared queue.

The figure 3 shows the process of selecting a to-place VM. This process is contained in the QueueMonitor service. It is an infinite loop that always looks for new VMs to place. First, it checks if the local queue contains virtual machines. If it is the case, it pops one and starts the placement process. If not, it checks if the shared queue contains to-place VMs. If it is the case, it adds a set of VMs taken from the shared queue to the local queue. Then, it goes back to the first step i.e. it checks if the local queue is empty. If the shared queue is empty, it randomly selects another Queue service. Then, it tries to steal a set of VMs from the selected Queue service. If the set is not empty, it adds it to the local queue and goes back to the first step. If not, it goes back to the step where it randomly selects another Queue service.

The placement function is presented in the figure 4. Its goal is to create a set of HNs where the VM can fit. First, based on the user's objectives,

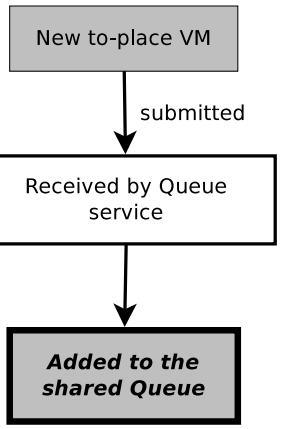

Figure 2: Sending the request to start a new Virtual Machine

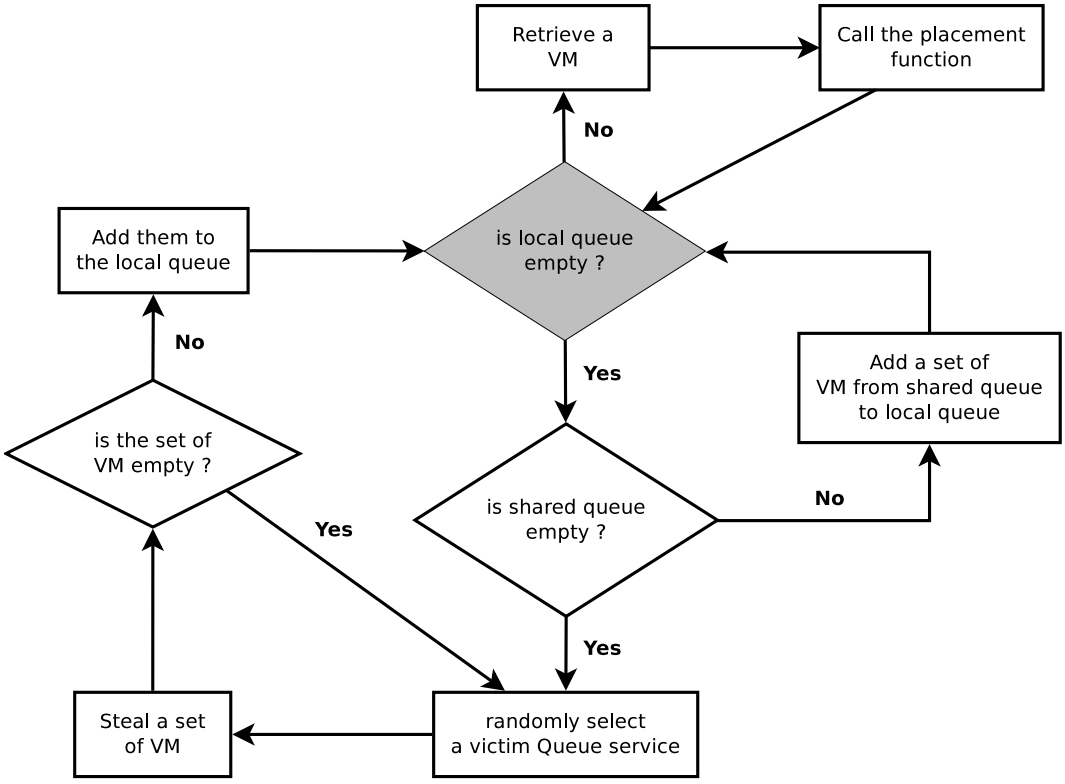

Figure 3: Function to retrieve a to-place VM

a score request is built. If the user's objectives do not specify a location or if they fit with the local datacenter, a score request is built for the HNs within the local datacenter i.e. the datacenter where the SAgent is running. If the user's objectives state that the local datacenter does not fit e.g. the user defined another country that the one of the current datacenter, a score request is built for a datacenter that fits the user's objectives. Then, the score request is sent through multicast i.e. through the MScore service. If at least one HN sent back its score (within 200ms), the election function is called. If no HN returns its score, the placement function tries to expand the score request. Next, if the election function was not able to place the VM, the placement function tries to expand the score request.

If possible, the score request is expanded to a larger geographic location i.e. if the user's objectives allow it or if the score request does not already span on the whole cloud. For example, if the score request targets one datacenter and the user's objectives sets a specific country, the score request is rewritten to target not only a datacenter but all the datacenters within the given country. Then, the new score request is sent. If it is not possible to expand the score request, the VM placement failed. A failure "to place the VM" event is built and sent to the Monitor service. Thus, the user can know that his VM has not been started and why.

The placement function calls the election function with the parameter $depth = 0$. This parameter will limit the number of recursive migrations (see section 8). Finally, if the placement succeed i.e. if the election function has placed the VM, a successful "to place the VM" event is sent to the Monitor service. Thus, the user can know that his VM has been started and where. Accordingly, it ables the user to verify if his objectives have been respected.

The election function is described in the figure 5. Based on the list of HNs that have returned their score in the placement function, a HN is elected i.e. the one with the highest score. Then, through the LaunchVM service, the HN is asked to start the VM. At the same time, a counter, n, is initialized to 0. This counter will limit (to 3) the number of VMs that can be migrated to free resources for this new VM. Without the counter, an infinite loop can happen. If there is enough resources available, the VM is started.

If there is not enough resources on the HN, it tries to free resources by migrating one (or multiple) virtual machine(s). But before, it checks if n is smaller than 3. If it is not the case, the VM placement has failed. Else, it increments n and creates a second counter m initialized to 0. m limits the number of failed migration to 3. As n, m protects against an infinite loop. First, it elects a virtual machine to migrate (this election process is described in the section 6.3) then tries to migrate it. To migrate the VM, it

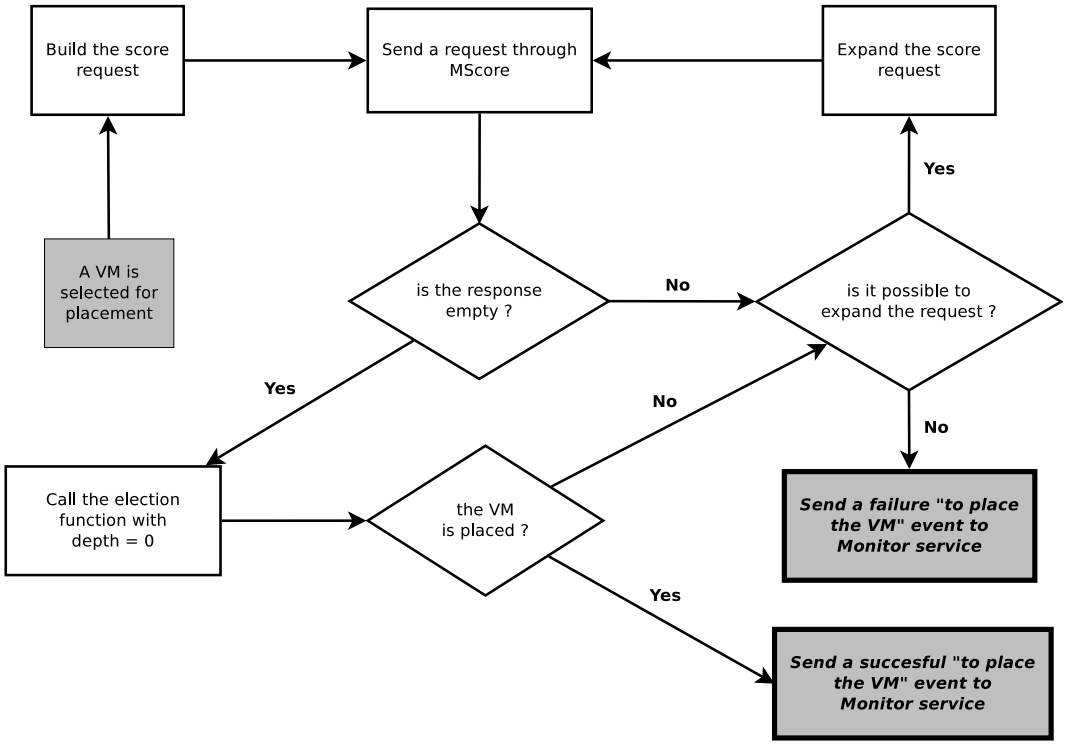

Figure 4: Function to retrieve a set of HN where to place the VM

calls the Migrate function (described in the section 8) with two parameters *depth* and *vm*, the elected VM. At the same time, m is incremented. If it fails, it checks if m is smaller than 4. If it is the case, it elects a new VM and tries to migrate it. Else, the VM placement fails.

If the migration successes, the function goes back to check if enough resources are available on the HN to start the VM for which the placement function has been invoked.

8 Reconfiguration

In this section, we describe how the reconfiguration process takes place. The reconfiguration function described in the figure 6 is invoked when a migration event is received. First, the function checks if the event contains a targeted VM i.e. a VM to migrate. If the event contains no VM, it elects one and set the parameter *depth* to 0 and adds the elected VM into the list $prevVM$.

If not, it checks if the *depth* parameter is smaller than 3. *depth* allows to limit the number of recursive migration. A recursive migration happens when we need to migrate a VM to migrate another VM. If we were not able to limit the number of recursive migration, a complete migration process could span on hundreds of VMs and HNs. Thus, the migration process will consume most of the resources of the cloud and deteriorates the performance of a large set of VMs.

If *depth* is smaller than 3 or when a VM is elected, *depth* is incremented. Then using the same approach than the one described in the figure 4, a list of HNs that can run the VM is built. Then the migration function is called to migrate the VM. If the migration succeeds, a successful "to migrate the VM" event is sent. Else, the score request is expanded and the function goes back to build a list of HNs.

The migration function shown in the figure 7 is similar to the placement function described in the figure 5 except it is dedicated to migrate a selected VM to a list of HNs. As we explain in the section 6.3, the protection against cyclic migration is done through the VM score algorithm. Thus, we do not need to add anything against it in our migration function.

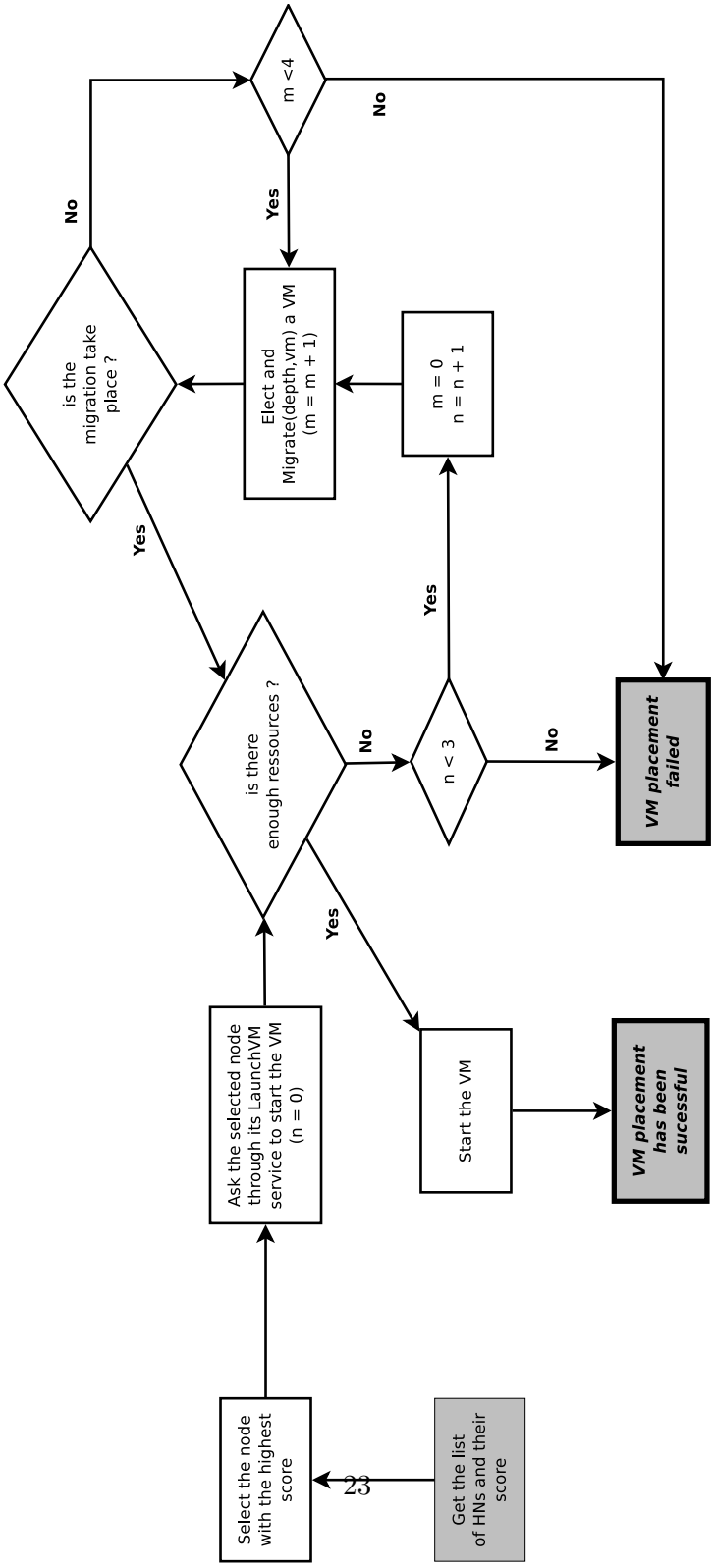

Figure 5: Function to elect a HN for a VM and place it

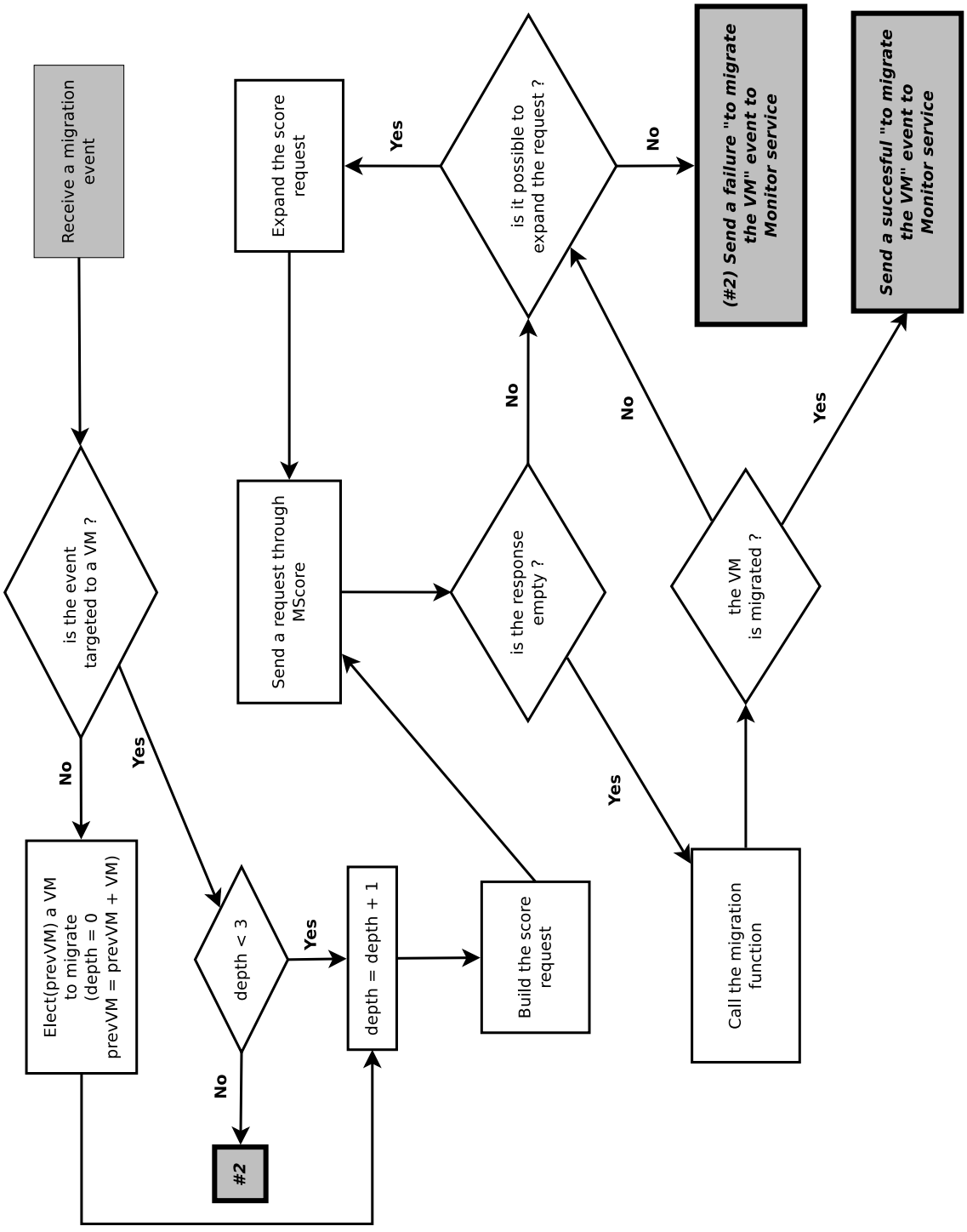

Figure 6: Function to reconfigure a HN based on a migration event

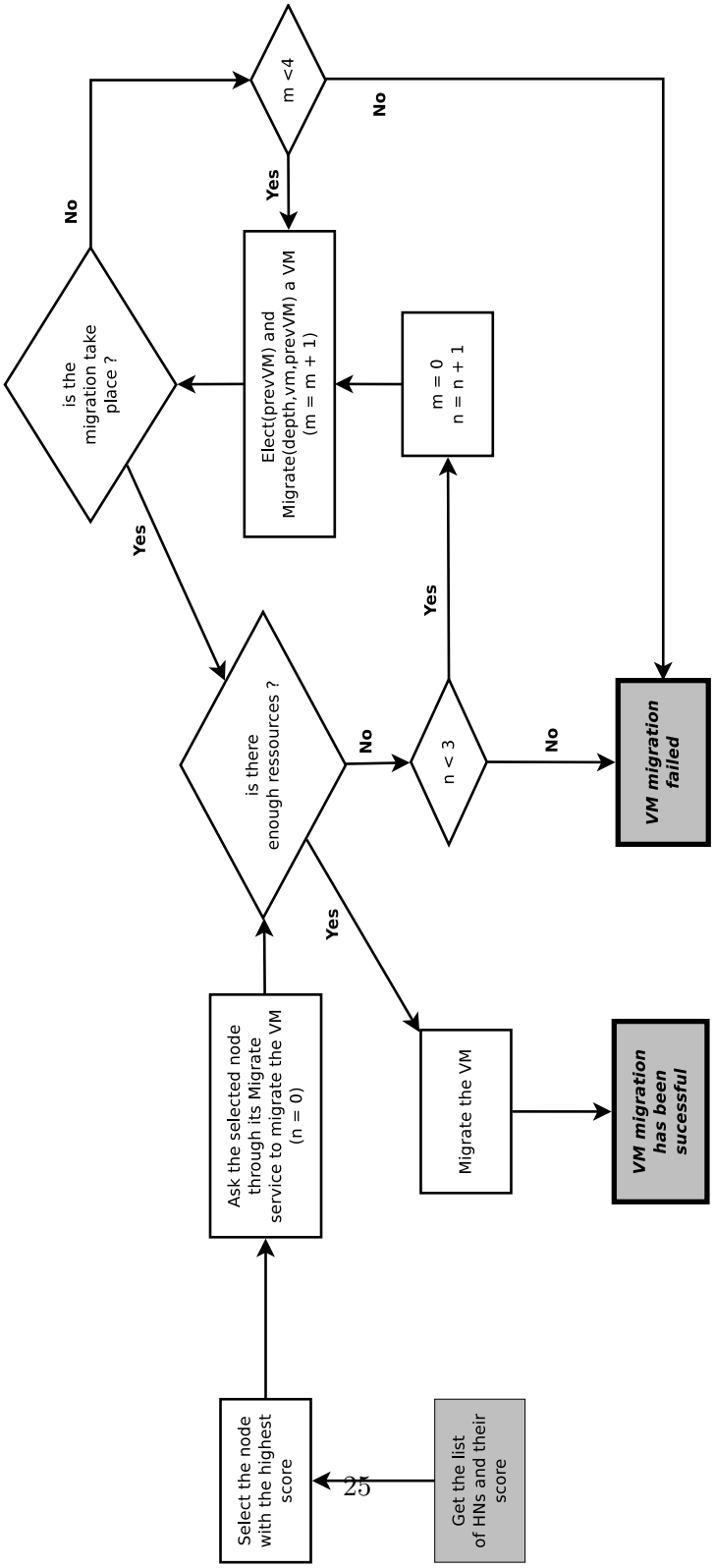

Figure 7: Function to elect a HN for a VM and migrate it

9 Evaluation

9.1 Simulation

Our architecture is implemented on top of the Java based library JXTA [11] because of its capabilities of services discovery and multi-cast communication. To evaluate our cloud scheduler, the VM, HN and SN are simulated by a Java thread. For the VM, we simulate the resources' consumption by adding or reducing the amount of resources used by the VM. This amount is randomly peak between 0 and the soft quota of the VM. The time lapse between adding and reducing the resources is randomly select between 10ms and 1s. The HN are simulated by three different resources scenario presented in the table 1.

	Nb of Core	**Mhz**	**RAM in Gb**
Host_1	4	3.2	16
Host_2	8	3	32
Host_3	16	3	64

Table 1: HN Resources Configuration

We have created a simulated cloud presented in the table 2. This cloud is composed of 6 different datacenters in 5 countries. Three countries are within Europe and three from America. Each datacenter has a set of HNs and one SN. The HNs have different capabilities as exposed in the table 2.

Continent	**Country**	**Datacenter**	**Tier**	**Hypervisor Node**		
				Quantity	***Capabilities***	***Config***
AM	USA	SF	3	10	PCIDSS, ISO-27001	Host_2
				50	HPC, HPN, GPU	Host_3
				100		Host_1
		NY	4	50	PCIDSS, ISO-27001	Host_2
				50	HPC, HPN, GPU	Host_3
				100		Host_1
	CA	OT	2	200		Host_1
EU	FR	PA	4	45	PCIDSS, ISO-27001	Host_2
				50	HPC, HPN, GPU	Host_3
				70		Host_1
	NL	AM	4	150	HPC, HPN, GPU	Host_3
				100		Host_1
	HU	BU	1	100		Host_1

Table 2: HN Resources Configuration

We try different VM scenario:

1. A web site: 3 VMs that require 1x2Ghz CPU and 2Gb of RAM. There is no user's objective.

2. A social web site: 70 VMs that require 1x2Ghz CPU and 2Gb of RAM. There is no user's objective.

3. A card payment software: 20 VMs that require 2x2Ghz CPU and 2Gb of RAM. The user's objectives for them are a PCIDSS, ISO-27001 and Tier 4 grade datacenter.

4. A critical data mining software: 30 VMs that require 4x2Ghz and 4Gb of RAM. The user's objectives for them are a Tier 3 datacenter. Moreover, all the VMs need to be located within Europe and no VM from other users must run on the same hardware.

5. An image rendering software: 40 VMs that require 2x2.5Ghz and 4Gb of RAM. There is no user's objective.

6. A scientific simulation HPC cluster: 40 VMs that require 4x3Ghz and 8Gb of RAM. The HN must provide HPC, HPN and GPU capabilities. Moreover, all the VMs must be allocated within the same datacenter.

7. An industrial simulation HPC cluster: 30 VMs that require 4x3Ghz and 8Gb of RAM. The HN must provide HPC, HPN and GPU capabilities. Moreover, all the VMs must be allocated within the same European datacenter and no VM from other users must run on the same hardware.

8. A HPC grid: 200 VMs that require 4x3Ghz and 8Gb of RAM. The HN must provide HPC, HPN and GPU capabilities. The grid can span on multiple datacenter around the world with at least 50 VMs in Europe and 50 in America.

In our simulation, we uses the following static parameters for the score algorithm:

- $\alpha = 1$, $\beta = 1$: to give the same importance to the soft quota of processor or memory.

- $\gamma = 0.75$: because we want to give more weight to the virtual machine resources quota than the static resources on the node.

- $\delta = 1$: to give the same importance to the amount of free processor or free memory.

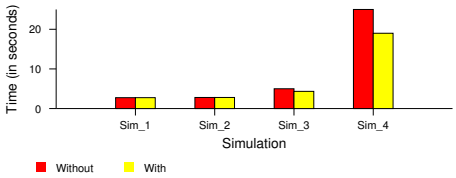

Figure 8: Amount of time for the placement of a new Virtual Machine

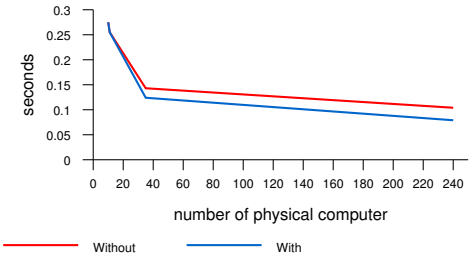

Figure 9: Amount of time per computer for the placement of a new Virtual Machine

9.2 Results

As shown on the figure 8 and 9, previous experimentation with an increasing number of HN as shown that the amount of time taken by the startup process for a virtual machine grows linearly with the number of nodes on the cloud. Indeed, with large size distributed systems, most of the interactions between agents can be parallelized. When a larger number of nodes are part of the cloud, more interactions can be done in parallel on different agents and thus the overall throughput increases i.e. the number of VM placement in a given time grows. Moreover, most of the placement and reconfiguration take place in one datacenter or one country thus limiting the scope of the score request.

We compute the average time between the detection of an overloaded HN and its solving. Each simulation was kept running for one day to evaluate if the placement of virtual machines was "good" enough. 15 migrations for the 1,000 virtual machines has been required and they took 19 seconds each. Consequently, we state that the placement of virtual machines on the distributed system is sufficiently "good" because the number of migrations is low.

With cooperative schedulers, both placement and migration processes are speed up as shown on the figures 8 and 9. It can be done because a vast majority of services can be loosely coupled.

Using the VM scenario presented in the section 9.1, we have computed preliminary results: an average deployment time for each one (figure 10) and an average deployment time per VM for each one (figure 11). This time does not take into account the actual deployment time but just the time to place the VM. As it is loosely coupled, the deployment of multiple VMs

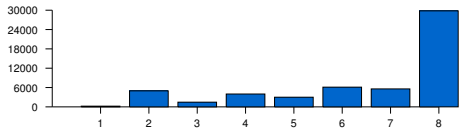

Figure 10: Deployment time for each VM scenario

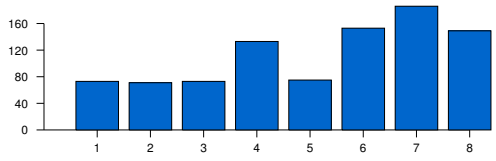

Figure 11: Deployment time per VM for each VM scenario

can be done at the same time. Three objectives impact the VM placement time and thus the VM scenario deployment time: nearVM, alwaysAlone and different location within the same scenario.

9.3 Traceability

As we previously state, traceability within cloud allows an user to check if his objectives have been respected. Moreover, it helps to increase the user's trust to the cloud's infrastructure as he can verify the enforcement of his objectives by himself. Thus, we have created monitoring services that allow a user to extract information about the placement and reconfiguration of his VMs.

For example, in the Listing 9.3, an user has received a report about one of his VM placement process. The first line registers the event representing the user request to start a new VM. The second line displays that the VM has been added in a scheduler queue. The third shows that the scheduler is trying to place the VM. Finally, the fourth line shows that the VM has been started and on which HN it is placed. The fifth registers that the whole process has finished without encoring any issues. Using the HN name, the user can check that his objectives have been respected. But, these traces do not show where the scheduling happens as it is not useful for the user and can leak informations about the inner architecture of the cloud.

```
10/2/2011 12:43:0 - Asking to add a VM Event: VM #1 is sent to the cloud.
10/2/2011 12:43:0 - Add VM to Queue Event: VM #1 added to the queue
10/2/2011 12:43:0 - Try to Place VM Event: Trying to place VM #1
10/2/2011 12:43:0 - Adding a VM Event: VM #1 has been added on LD_PEER_4
10/2/2011 12:43:0 - Have Place VM Event: VM #1 has been placed.
```

Other traces like the one in the Listing 9.3 show the inner working of the scheduling infrastructure. The two lines describe a scheduler stealing to-place VMs from another scheduler.

```
9/2/2011 20:30:0 - SAgent_1 - Have stolen VM(s) Event: 1 VM(s).
9/2/2011 20:30:0 - SAgent_2 - VM(s) have been stolen Event:
                        Too much work: someone stole me 1 VM(s).
```

Finally, the trace in the Listing 9.3 shows how the cloud manages an overloaded HN. First, the overloaded HN is detected. Then, it elects a VM and asks other HNs their score. At the third line, when a HN is elected and it can host the VM, the migration is allowed. The migration itself is divided into two part, one representing the suspension of the VM on the overloaded HN (LD_PEER_2) and one representing the resume of the VM on the new HN (LD_PEER_3).

```
26/1/2011 0:4:10 - Overload Event: LD_PEER_2 is overloaded
26/1/2011 0:4:10 - Asking Migration Event: 32 migration asked to LD_PEER_2
26/1/2011 0:4:11 - Allowed Migration Event: 32 has migrated to LD_PEER_3
26/1/2011 0:4:11 - Removing a VM Event: VM #32 has been deleted on LD_PEER_3
26/1/2011 0:4:11 - Adding a VM Event: VM #32 has been added on LD_PEER_2
```

10 Conclusion

Our paper presents a novel scheduler dedicated for VM placement and reconfiguration on clouds. It is based on P2P architecture allowing a fully decentralized model. The scheduling decisions are based on dynamic-resources centric algorithm that computes a score and on user's objectives. The score is based on both static and dynamic resources usage of processor and memory but also uses the resources quota associated with each VM. The user's objectives eases the expression of quality of trust and protection and the deployment of HPC architecture. These objectives are given by the user for each of his VMs.

Moreover, the scheduler agents can cooperate together through a distributed and decentralized facility. We have implemented our model using Java and JXTA. Then, we implement it within our cloud testbed to evaluate our model efficiency. As we show, the simulation results are encouraging. Indeed, we do not see any scalability bottleneck and the balancing of resources works great. Furthermore, the user's objectives eases the deployment and the expression of user's goals without impacting too much the scheduling time.

Future works will tackle the over-migration issue when a VM is migrating constantly from a node to another. Indeed, this type of issue can lead to a major overhead for the migrating VM but also increases the load on agents. The fault tolerance of the stealing algorithm is still an open question for our work stealing algorithm. Moreover, we want to do extensive experiments of our architecture and on even larger scale than the one proposed here. Another future work will be to test our solution on a real cloud. We have already implemented libvirt on our solution, thus, bringing the support for all mainstream hypervisors.

References

[1] Mauro Andreolini, Sara Casolari, Michele Colajanni, and Michele Messori. Dynamic load management of virtual machines in cloud architectures. In *Cloud Computing*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 201–214. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-12636-914.

[2] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*, pages 119–128. IEEE, 2007.

[3] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2):141–154, February 1988.

[4] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[5] Marco Comuzzi, Constantinos Kotsokalis, George Spanoudakis, and Ramin Yahyapour. Establishing and monitoring slas in complex service based systems. *Web Services, IEEE International Conference on*, 0:783–790, 2009.

[6] J. Dinan, S. Olivier, G. Sabin, J. Prins, P. Sadayappan, and C.-W. Tseng. Dynamic load balancing of unbalanced computations using message passing. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, March 2007.

[7] James Dinan, D. Brian Larkins, P. Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. Scalable work stealing. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–11, New York, NY, USA, 2009. ACM.

[8] H. El-Rewini, T.G. Lewis, and H.H. Ali. Task scheduling in parallel and distributed systems. *Prentice-Hall, Inc. Upper Saddle River, NJ, USA*, page 290, 1994.

[9] Eugen Feller, Louis Rilling, Christine Morin, Renaud Lottiaux, and Daniel Leprince. Snooze: A scalable, fault-tolerant and distributed consolidation manager for large-scale clusters. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, pages 125–132, Washington, DC, USA, 2010. IEEE Computer Society.

[10] Ian Foster and Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *In 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03*, pages 118–128, 2003.

[11] Li Gong. Jxta: a network programming environment. *Internet Computing, IEEE*, 5(3):88–95, May/Jun 2001.

[12] James A. Hall and Stephen L. Liedtka. The sarbanes-oxley act: implications for large-scale it outsourcing. *Commun. ACM*, 50:95–100, March 2007.

[13] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 41–50, New York, NY, USA, 2009. ACM.

[14] Wei Huang, Qi Gao, Jiuxing Liu, and Dhabaleswar K. Panda. High performance virtual machine migration with rdma over modern interconnects. In *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, CLUSTER '07, pages 11–20, Washington, DC, USA, 2007. IEEE Computer Society.

[15] Khaled M. Khan and Qutaibah Malluhi. Establishing trust in cloud computing. *IT Professional*, 12:20–27, September 2010.

[16] Tomas Koutny and Jiri Safarik. Load redistribution in heterogeneous systems. *Autonomic and Autonomous Systems, International Conference on*, 0:24, 2007.

[17] Keith W. Miller, Jeffrey Voas, and Phil Laplante. In trust we trust. *Computer*, 43:85–87, October 2010.

[18] M. Mowbray. The fog over the grimpen mire: cloud computing and the law. *Script-ed Journal of Law, Technology and Society*, 6(1), 2009.

[19] Yoshitomo Murata, Hiroyuki Takizawa, Tsutomu Inaba, and Hiroaki Kobayashi. A distributed and cooperative load balancing mechanism for large-scale p2p systems. In *SAINT-W '06: Proceedings of the International Symposium on Applications on Internet Workshops*, pages 126–129, Washington, DC, USA, 2006. IEEE Computer Society.

[20] Siani Pearson and Azzedine Benameur. Privacy, security and trust issues arising from cloud computing. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 693–702, Washington, DC, USA, 2010. IEEE Computer Society.

[21] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[22] Jonathan Rouzaud Cornabas. A Distributed and Collaborative Dynamic Load Balancer for Virtual Machine. In *5th Workshop on Virtualization in High-Performance Cloud Computing (VHPC '10) Euro-Par 2010*, page _, Ischia, Naples Italy, 08 2010. Lecture Notes in Computer Science series. 10 pages.

[23] J. Salmon. Clouded in uncertainty–the legal pitfalls of cloud computing. *Computing*, 24, 2008.

[24] Ravi Sandhu, Raj Boppana, Ram Krishnan, Jeff Reich, Todd Wolff, and Josh Zachry. Towards a discipline of mission-aware cloud computing. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, CCSW '10, pages 13–18, New York, NY, USA, 2010. ACM.

[25] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *IEEE Transactions on Networking*, 11, February 2003.

[26] Franco Travostino. Seamless live migration of virtual machines over the man/wan. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[27] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology - Volume 02*, CIT '09, pages 357–362, Washington, DC, USA, 2009. IEEE Computer Society.

[28] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. Sla-aware virtual resource management for cloud infrastructures. *Computer and Information Technology, International Conference on*, 1:357–362, 2009.

[29] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923 – 2938, 2009. Virtualized Data Centers.