Image Transfer and Storage Cost Aware Brokering Strat. for Multiple Clouds

Frédéric Desprez, Jose-Luis Lucas-Simarro, Rafael Moreno Vozmediano, Jonathan Rouzaud-Cornabas

Image Transfer and Storage Cost Aware Brokering Strat. for Multiple Clouds

F. Desprez, J. L. Lucas-Simarro, R. Moreno-Vozmediano, J. Rouzaud-Cornabas

RESEARCH REPORT

N° 8445

November 2013

Project-Team Avalon

ISSN 0249-6399 ISRN INRIA/RR--8445--FR+ENG

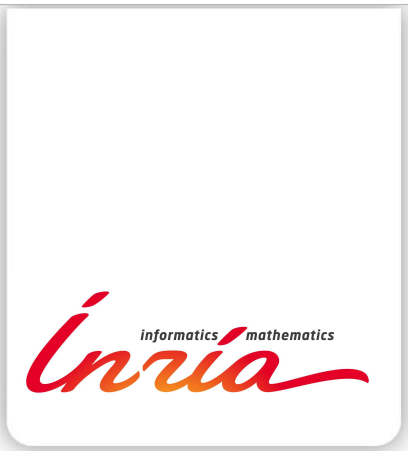


Image Transfer and Storage Cost Aware Brokering Strat. for Multiple Clouds

F. Desprez*, J. L. Lucas-Simarro†, R. Moreno-Vozmediano†, J. Rouzaud-Cornabas*

Project-Team Avalon

Research Report n° 8445 — November 2013 — 15 pages

Abstract: Nowadays, Clouds are used for hosting a large range of services. But between different Cloud Service Providers, the pricing model and the price of individual resources are very different. Furthermore hosting a service in one Cloud is the major cause of service outage. To increase resiliency and minimize the monetary cost of running a service, it becomes mandatory to span it between different Clouds. Moreover, due to dynamicity of both the service and Clouds, it could be required to migrate a service at run time. Accordingly, this ability must be integrated into the multi-Cloud resource manager, *i.e.* the Cloud broker. But, when migrating a VM to a new Cloud Service Provider, the VM disk image must be migrated too. Accordingly, data storage and transfer must be taken into account when choosing if and where an application will be migrated. In this paper, we extend a cost-optimization algorithm to take into account storage costs to approximate the optimal placement of a service. The data storage management consists in taking two decisions: where to upload an image, and keep it on-line during the experiment lifetime or delete it when unused. Although the default approach can be to upload an image on demand and delete it when it is no more used, we demonstrate that by adopting other policies the user can achieve better economical results.

Key-words: Cloud Brokering, Resource Allocation, Storage, Data Transfer, SimGrid Cloud Broker

* INRIA, LIP ENS Lyon, France, Email: FirstName.LastName@inria.fr
† Dept. de Arquitectura de Computadores y Automatica, Universidad Complutense de Madrid, Madrid, Spain

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Transfert d'images et algorithmes de gestion de ressources prenant en compte les coûts pour les fédérations de clouds

Résumé : De nos jours, les Clouds sont utilisés pour héberger un grand ensemble de services. Mais entre les différents fournisseurs de service Cloud, les modéles de prix et le prix de chaque ressource sont très différents. De plus, héberger un service dans un unique Cloud est une des causes principales d'interruption de service. Pour améliorer la résistance et diminuer le coût monétaire d'une application, il devient obligatoire de la distribuer dans plusieurs Clouds. En outre, à cause de la dynamicité de l'application et des Clouds, il peut être nécessaire de migrer l'application pendant l'exécution. Par conséquence, cette capacité doit être intégrée dans le gestionnaire de ressources multi-Cloud i.e. le Cloud Broker. Mais, quand une VM migre vers un nouveau fournisseur de service Cloud, l'image disque de la VM doit être migrée également. Par conséquence, le stockage et transfert de donnée doivent être pris en compte quand il est choisi si une application doit migrer et où.

Dans ce papier, nous étendons un algorithme d'optimisation de coût pour prendre en compte le coût du stockage afin d'approximer le placement optimal d'une application. La gestion du stockage de donnée consiste à devoir prendre 2 décisions: où l'image doit être envoyée et doit-elle être conservée ou supprimée quand elle n'est plus utilisée. Même si l'approche par défaut peut être d'envoyer l'image à la demande et la supprimer quand elle n'est plus utilisée, nous démontrons qu'en adoptant d'autres politiques l'utilisateur peut réussir à atteindre de meilleurs résultats économiques.

Mots-clés : Cloud Brokering, Resource Allocation, Storage, Data Transfer, SimGrid Cloud Broker

1 Introduction

The use of Cloud computing technology has gained popularity in recent years both in industry and academia [2]. Nowadays, many companies are moving their business to the Cloud, by deploying their services and executing their workloads in private or public Clouds, following their particular business models. Cloud platforms are more and more used for the deployment and execution of service-based applications that consist of different components providing business services. However, the growing number of Cloud Service Providers (CSPs) has created a diverse Cloud market with different pricing models, different prices within the same pricing model, different instance types, or different agreement conditions. To serve as intermediary between end users and CSPs, Cloud brokers emerged as a powerful tool [3, 8]. A Cloud broker can help Cloud users to choose the right virtual machine (VM) placement when deploying their services across multiple Clouds, also allowing them to switch between providers in order to always get the best conditions.

When moving from a single Cloud to a multi-Cloud environment, migrating a service from a Cloud to another becomes a basic operation. In previous works [11], migrating a service (and the related VMs) was modeled as the termination of VM(s) in the source CSP and the startup of VM(s) in the destination CSP. But this model just addresses the case of using CSP's pre-defined images, which are available in every commercial Cloud. In case of customized images, before starting a VM, it is required to transfer its Virtual Machine Image (VMI) to the Cloud. Accordingly, when migrating a VM from a CSP to another one, it is required to upload the VMI to the destination CSP. Transferring and storing VMI has a monetary cost on top of the time it takes to do the transfer. Consequently, when a Cloud broker chooses a VM placement, it must take into account the management of VMI, otherwise it could take improper decisions. For example, the cost of transferring and storing a VMI in a new Cloud can be greater than the cost saved by migrating to this new Cloud.

From the beginning of the Cloud technology adoption, Amazon EC2 [1] has been consolidated as one of the major Infrastructure as a Service (IaaS) CSPs. Amazon provides users with a wide range of possibilities for the deployment of their infrastructure, in terms of the instance types or pricing schemes. Moreover, Amazon offers different regions which users can envisage as different Clouds with different conditions, such as price or performance. Furthermore, Amazon was the first CSP to offer Spot Instances (SIs) to sell their cluster's spare capacity. Spot prices change dynamically according to free capacity and actual demand, which opens a new challenge for price-aware Cloud brokers. Spot instances requested with bid price higher than or equal the current spot price will be served, but otherwise, the provider will terminate out-of-bid instances abruptly.

Nowadays, with the inclusion of Cloud brokers to help users to process Cloud market information, the innovation on Cloud brokering algorithms is an open research issue. The proposal of new placement algorithms and policies in the broker for optimal deployments of virtual services among multiple Clouds, based on different optimization criteria (cost optimization, performance optimization, energy efficiency, etc.) is critical. Moreover, several research works have studied how to take advantage from the different Cloud features in case when provider and user conditions remain unchanged [15] or change along time [11].

In this paper, we propose a Cloud brokering algorithm improvement that

optimizes the infrastructure deployment cost considering different data storage and transfer policies. We propose several storage policies and we include the algorithm in the SimGrid Cloud Broker [7], a simulation tool for multi-Cloud environments. Finally, we test the policies' behavior using simulation, and select one as the best policy for Cloud brokering.

The rest of this paper is organized as follows: Related work is discussed in Section 2. Section 3 describes the problem, while the system architecture, and the detailed design of the proposed solution are explained in Section 4. The experimental environment, conditions, and the results are described in Section 5. Finally, Section 6 contains the conclusions and future directions for this work.

2 Related Work

This work is an extension of [11], where the authors explored the Cloud brokering issue applied to dynamic scenarios, especially when pricing conditions change along time. In this previous work, the placement action was done periodically in order to reallocate the virtual infrastructure to the best fitting Clouds. Hence, the concept of performance degradation as a placement constraint due to the periodic reallocation action is introduced. Moreover, a comparison is made between static and dynamic deployments showing the cost improvement potential of the use of brokering mechanisms. However, in this work, they do not take into account the image storage cost neither than the image transfer cost.

Two key concepts to take into account when using Cloud brokering algorithms are data checkpointing and migration. How to improve both techniques and how to combine them are still an open challenge. In [19], the authors investigate several approaches to reduce monetary cost and task completion time using migration and checkpointing techniques with Amazon EC2 spot instances. However, they assume negligible storage cost of checkpointing and also negligible migration costs. There are also some efforts in minimizing migration disadvantages from clusters to Clouds, *i.e.* migration periods or application downtimes. At first, these efforts showed solutions to improve virtual machine migration inside clusters [6]. Later, the researchers focused their investigation on Cloud environments, where physical clusters are geographically distributed. For example, Travostino et al. in [16] migrate virtual machines on a WAN area with just 1-2 seconds of application downtime.

Simulators are often seen as a technical validation mean, and most of the time, a custom tool is developed for each paper. The problem is that most simulators have not gone through a proper validation. Accordingly, their accuracy and the reproducibility of their results have not been verified. Cloud simulators are essential to test both Cloud brokering algorithms and VM migration techniques. There are different simulation tools that can be used for the experimental development of Cloud infrastructures, such as CloudSim, SGCB, or iCanCloud. CloudSim [4, 9] is an extensible toolkit used to model and simulate Cloud infrastructures, including data centers, users, user workloads, and application provisioning. CloudAnalyst [18] is built on top of the CloudSim toolkit; it provides visual modeling, easy to use graphical user interfaces, and large-scale application simulations deployed on the Cloud infrastructure. Application developers can use CloudAnalyst to determine the best approaches for allocating resources among available data centers to serve specific requests and determine

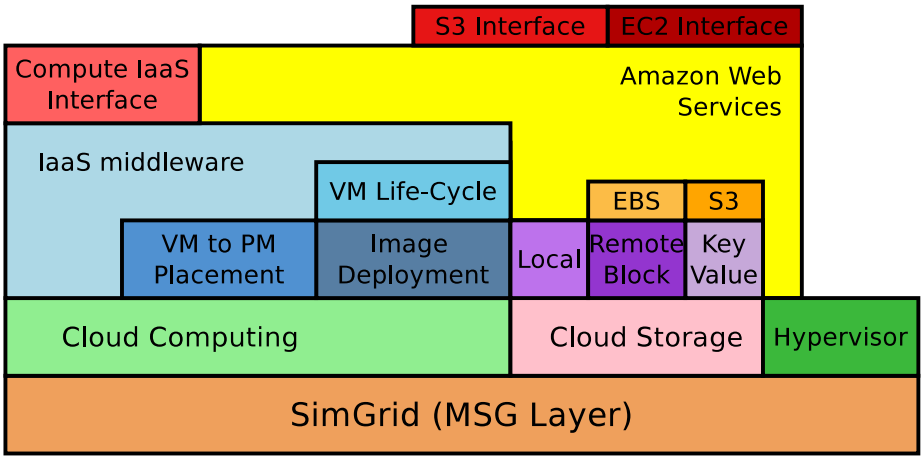


Figure 1: SimGrid Cloud Broker architecture.

the related costs for these operations. But as shown in [14], CloudSim is not scalable and lacks a proper validation process for its network model. Alberto Nuñez et al. [13] introduce a novel simulator of Cloud infrastructures called iCanCloud. It reproduces the instance types provided by a given CSP, and contains a user-friendly GUI for configuring and launching simulations, that goes from a single VM to large Cloud computing systems composed of thousands of machines. But they simulate a Cloud as viewed by a CSP and not by an Cloud user. Furthermore, they lack a proper validation process for their models and show a limited scalability in term of number of VMs.

To avoid accuracy and reproducibility issues, SimGrid Cloud Broker (SGCB) [7] has been implemented on top of SimGrid [5]. SimGrid is an open-source, generic distributed systems simulation framework providing very realistic and flexible simulation capabilities. SimGrid was conceived as a scientific instrument and the validity of its analytical models were thoughtfully studied [17]. It has been used as simulator in 119 scientific publications[1]. SGCB provides researchers and engineers with a multi-Clouds environment to evaluate their provisioning and elastic algorithms and full applications through simulation before running them on a full fledge Cloud. Furthermore, SGCB provides a simulation of a Cloud and multiple Clouds as seen by a Cloud User and not as seen by the CSP as other simulators do. As most of the interactions between a Cloud and an user is done through an API interface, SGCB reproduces the same approach. Moreover AWS is the most known and used Cloud and the related APIs (EC2 and S3) are defacto standards. SGCB replicates them to ease the transition toward AWS or other Clouds providing the APIs. Figure 1 describes the inner-architecture of SGCB and all its different components. In this paper, we used SGCB because it is the best fitting simulator for our case (multi-Clouds, Cloud user point of view, validated models). Moreover, we use SGCB as it and did not modify its inner-architecture. We have just used the S3 and EC2 interface as we will have done on the real AWS platform.

[1] http://simgrid.gforge.inria.fr/Publications.html

3 Problem Description

In this work we explore the convenience of using brokering mechanisms to reallocate part or the entire infrastructure to another placement not only taking into account compute cost, but also image storage cost. Although this is a multi-Clouds challenge, in the evaluation section, we consider only Amazon EC2 to simulate different Clouds by using its regions as particular isolated Clouds. Other Cloud infrastructures can easily be added to our simulator, provided that we have an accurate model of their costs and overall architecture.

Amazon offers the following two types of storage:

- Elastic Block Storage (EBS), which is designed specifically for EC2 instances, and which allows users to create remote block storage volumes that can be mounted as devices by EC2 instances.
- Simple Storage Service (S3), which provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It is billed in GB of data per month, and there are some billing intervals with which Amazon bills less $ per GB as more amount of data is consumed.

Due to annual failure estimations, EBS users should keep an up-to-date snapshot on S3, or have a backup of the contents somewhere else that they can restore quickly enough to meet their needs in the case of a failure. On the other hand, S3 is subject to eventual consistency, which means that there may be a delay in writes appearing in the system whereas EBS has no consistency delays. Also EBS can only be accessed by one machine at a time whereas snapshots on S3 can be shared between different VMs. EBS volumes can only be accessed from an EC2 instance in the same availability zone whereas snapshots on S3 can be accessed from any availability zone in one region.

In previous works [11, 12] we have taken the EBS option (launching predefined instances and contextualizing them at boot time) in order to avoid storage costs at brokering level. However, we did not consider storage failures neither user-made images. Our algorithms leverage dynamic spot prices and the combination of different instance types to build the required infrastructure while optimizing costs.

In this work, the challenge is to make our algorithms aware of S3 storage cost in order to allow users to deploy theirs infrastructures among multiple Clouds.

4 Proposal

Our proposal consists in adding storage price information to the decision algorithm process introduced in [12], and validate our algorithms on SimGrid Cloud Broker (SGCB).

Focusing on the algorithm design, we want to deploy a given number n of VMs, $v_1...v_n$, across the m available Clouds (here Amazon EC2 regions), $c_1...c_m$, using a given number instance types, $it_1...it_l$. As Amazon also offers Spot Instance (SI) prices that continuously change, we define t as any one-hour period we use for scheduling. We consider a integer programming formulation where $X_{i,j,k}(t) = 1$ if virtual machine i, which belongs to instance type j, is

placed on Cloud k ($1 \leq i, j, k \leq n, l, m$ respectively); 0 otherwise. As SGCB is coded in Java, we have selected Choco Constraint Programming Library [10] to develop the algorithm within the simulator.

We want to minimize the Total Cost of the Infrastructure (TIC), which is formulated as follows:

$$TIC(t,x) = \sum_{i}^{n} \sum_{j}^{l} \sum_{k}^{m} X_{i,j,k}(t) * C_{j,k}(t,x) \tag{1}$$

with

$$C_{j,k}(t,x) = P_{j,k}(t) * Sc_k(x) \tag{2}$$

where: $P_{j,k}$ refers to the price of an instance type j in a Cloud k under Amazon's SI pricing scheme; and $Sc_{x,k}$ refers to the price of storing x bytes in Cloud k.

Finally, in the rest of this work we use for our deployments several brokering constraints presented in [12]. They are defined as follow:

- **Performance constraint:**

$$Perf(t) \geq Perf_{min} \tag{3}$$

  This means that the virtual infrastructure has to reach a minimum performance at each moment t. This performance can be measured in several ways, such as number of CPU cores, FLOPS, GB of RAM, or using application-defined indicators. To show an example, we choose the virtual infrastructure's number of cores as a performance measuring unit, so

$$Perf(t) = \sum_{i,j,k} X_{i,j,k}(t) * Cores_j \tag{4}$$

  where $Cores_j$ is the number of CPU cores of instance type j.

- **Reallocation constraint:** It provides the possibility of reallocating only a certain number of CPU cores in each scheduling decision. It is useful when it is critical to keep part of the virtual infrastructure working without stop to guarantee a certain number of CPU cores working at any moment. Moreover, it allows the control of infrastructure performance degradation while saving some money by taking advantage of dynamic pricing.

$$R_{min}(t) \leq reallocation(t) \leq R_{max}(t) \tag{5}$$

  In Equation 5, R_{min} and R_{max} refer to the minimum and maximum number of CPU cores, respectively, that the broker can reallocate. Reallocation is defined as the difference between the last deployment performed and the next deployment to perform, in terms of number of CPU cores deployed in each Cloud. For that purpose, the Cloud broker compares the current placement of the VMs with the new one.

$$reallocation(t) = \frac{\sum_{n}^{i} \sum_{j}^{l} \sum_{k}^{m} Abs(X_{i,j,k}(t) - X_{i,j,k}(t-1)) * Cores_j}{2} \tag{6}$$

In Equation 6, the reallocation parameter is divided by 2 because only the number of CPU cores needed to start in a new Cloud is taken into account. In other words, Equation 6 means the number of CPU cores to move across Clouds.

- **Instance type constraint:** It provides the possibility of using only a certain type of VMs in each deployment. It is then for instance possible to specify to 'only use small instance type' or 'use all available instance types'. In Equation 7, it_{min} and it_{max} refer to the minimum and maximum percentage of the instance types to use in each deployment.

$$it_{min}(j) \leq \frac{\sum_i^n \sum_k^m X_{i,j,k}}{n} \leq it_{max}(j) \\ ,1 \leq j \leq l \tag{7}$$

- **Placement constraint:** It provides the possibility to maintain a certain number of VMs in each cloud placement. I can be used to express the requirement of 'using only one particular Cloud' or 'using every available Cloud'. In Equation 8, loc_{min} and loc_{max} refer to the minimum and maximum percentage of the virtual infrastructure to deploy in the selected providers.

$$loc_{min}(k) \leq \frac{\sum_i^n \sum_j^l X_{i,j,k}}{n} \leq loc_{max}(k) \\ ,1 \leq k \leq m \tag{8}$$

As previously explained, we propose four different policies for Virtual Machine Image (VMI). These policies can be divided into 3 sub-policies (uploading, transferring and deletion of VMI). The first sub-policies specify when a VMI is uploaded to a Cloud, the second one expresses the method selected to transfer the VMI and the third one describes when the VMI can be deleted from a Cloud. For the uploading policy, we propose two strategies. The first one is called `Everywhere` (E) and it specifies that the VMI must be uploaded into all the potentially used Clouds at the beginning. Accordingly, even if a Cloud is not used but is considered by the Cloud brokering algorithm, the VMI will be uploaded there. This strategy has a monetary cost as the VMI is stored in all Clouds during the whole life-time of the application. The second uploading strategy is called `On-Demand` (O) and it specifies that the VMI is uploaded to a Cloud only when the Cloud brokering algorithm has specified that at least one VM will be started there. Accordingly when a Cloud is not used, the VMI is not stored in it. But this approach requires to upload the VMI to a Cloud before being able to start a VM there. Therefore it can induce delay on the VM startup, *i.e.* it adds the uploading time of the VMI.

For the VMI transfer policy, we evaluate two different strategies to transfer the VMI to a Cloud. The first one (`Get`) specifies that the VMI is uploaded by the user to the Cloud. The second one (`Copy`) specifies that the VMI is uploaded once by the user to the first Cloud and then the VMI is copied from a Cloud to another one. The second strategy must able transfer the VMI faster as network links between Clouds are faster than the ones between the user and each Cloud.

But, in AWS at least, uploading data from a user to a region is free but copying between region has a cost.

For the deletion policy, we also propose two strategies. The first one is called `Never` (N) and it specifies that once a VMI is uploaded to a Cloud it will never be deleted, *i.e.* until the end of the application's life. Accordingly, even if no VM are running in a Cloud, the VMI is still stored on it. Therefore, this strategy has a monetary cost as the VMI is stored on a Cloud even when it is not used. The second deletion strategy is called `Always` (A) and it specifies that when there is no VM running on a Cloud, the VMI must be deleted there. Accordingly, this strategy allows to reduce the monetary cost by only storing the VMI where it is needed. But, as the `O` strategy for uploading the VMI, it can induce a delay on the VM startup as it could be required to reupload several times a VMI to the same Cloud.

5 Evaluation

The experiments have been done using SGCB [7]. We have used the full AWS platform with all regions (8) and instance types (10). Furthermore, the Cloud broker is connected to all the regions through a 10Mb/s network link. For all experiments, we have used the `random` Spot Instance price statistical distribution of SGCB and all other prices have been retrieved on the AWS website [2] on the 20th of July 2013. We want to evaluate the impact on our algorithms of 3 parameters: the VMI size (0.5, 1, 2, and 5 GB), the number of VMs required (2, 5, 10 VMs), and the number of cores required (2, 5, 10, 15 cores). For each triple of parameters, we test each combination of storage, transfer, and deletion policies, as introduced in Section 4. The aim of these experiments is to know what is the best combination of VMI storage-transfer-deletion policy for different scenarios, and also to know the advantage of using Cloud brokering mechanisms, if any.

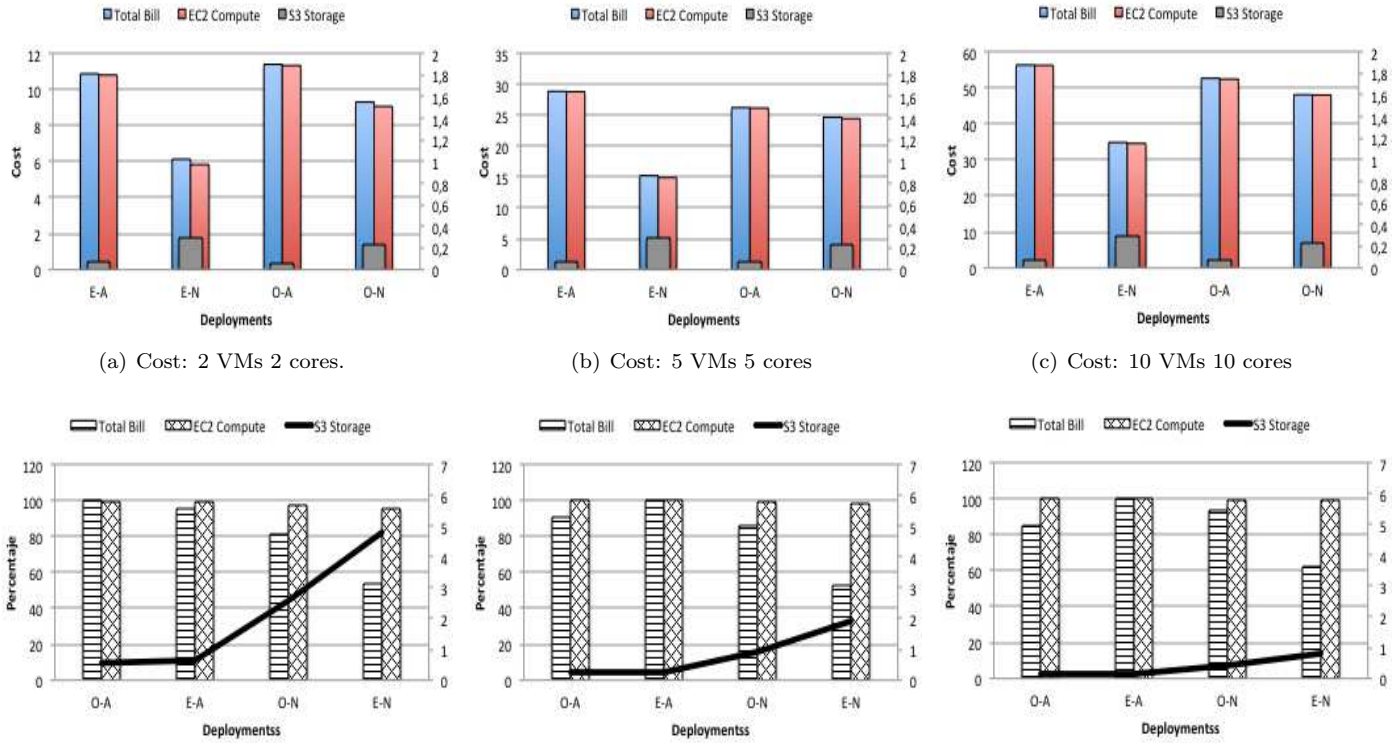
For the first set of experiments, we define the following constraints: 2 VMs with a least 2 cores, and using a 0.5 GB VMI. Furthermore, we always use the transfer policy `Get` at the moment. Obviously, the instance types used in this case are "small", since it is the only instance type that provides 1 core. Figure 2 shows the results of the simulation.

As one can see, the best combination of policies, in this case, is `Everywhere-Never` (`E-N`). In previous works [11, 12], we have considered EBS storage with VMI uploaded on-demand and deleted when not necessary, *i.e.* `O-A` policy. Indeed, our thoughts were not to pay for unused resources, i.e. uploading the VMI everywhere and never deleting them. But as it can be observed, in Figures 2(a), 2(b), and 2(c), with a S3 storage, it is clearly better to use the `E-N` policy.

These results can be explained as follows:

- `E-N` is the best solution because once the VMI has been uploaded in every Cloud at the beginning of the deployment, the algorithm is aware of the availability of a VMI in each Cloud, assumes its cost, and does not take into account the VMI upload cost in the deployment decision. Therefore,

[2] For EC2 http://aws.amazon.com/ec2/pricing/ and for S3 http://aws.amazon.com/s3/pricing/



(a) Cost: 2 VMs 2 cores.

(b) Cost: 5 VMs 5 cores

(c) Cost: 10 VMs 10 cores

(d) Percentage: 2 VMs 2 cores.

(e) Percentage: 5 VMs 5 cores.

(f) Percentage: 10 VMs 10 cores.

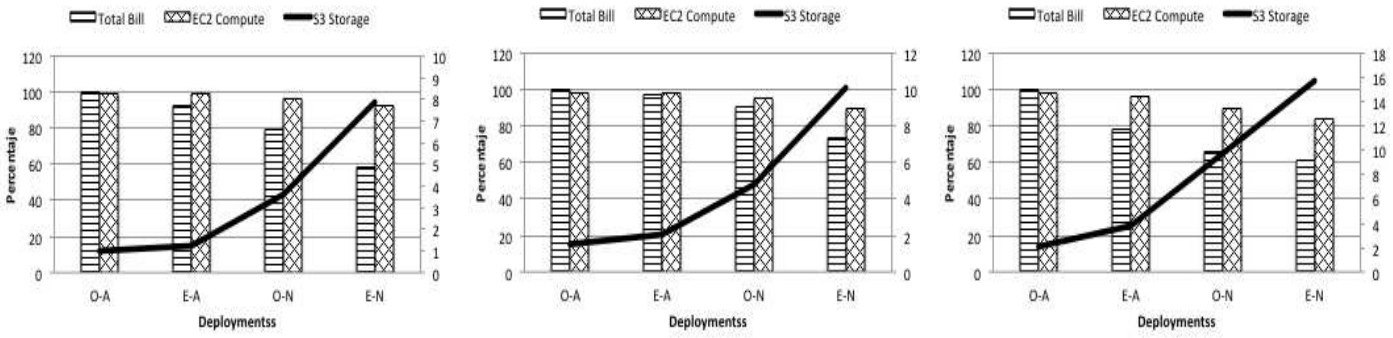
Figure 2: Simulation: 0.5GB of VMI.

the brokering algorithm focuses only on the best prices of each particular Cloud in time t, and gets full benefits of them by choosing the better one. Obviously, storage cost gets clearly incremented, but it is demonstrated that the total cost decreases in every case. Indeed, we can react much faster to changing prices as we do not have the delay of transferring the VMI to a new Cloud.

- `O-N` is not as good as the previous one, but its performance gets close to it, as long as the VMI is uploaded in more Clouds. Indeed, after some time the VMI tends to be stored in all Clouds because of the "never delete" policy, and the situation became similar to the aforementioned case (`E-N` policy). The decisions with this strategy get conditioned by the fact that the VMI is or is not already available in the Cloud with the best price. If not, the algorithm may reallocate the VM in another Cloud with a higher compute price, but lower cost considering the VMI upload.

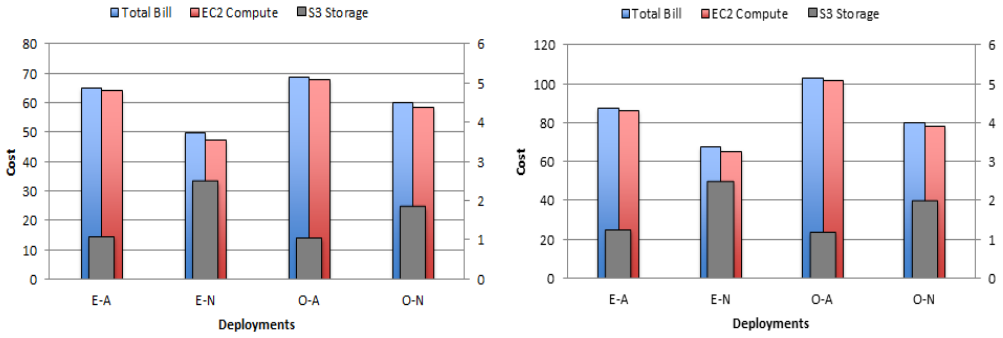
- For the other combined policies, `E-A` shows better behavior than `O-A`. This is because once the VMIs have been uploaded, the algorithm can choose the best placement in the first decision without worrying about storage costs. Once the first decision has been taken, the VMIs in the unused Clouds are deleted, and therefore the next decision will be taken in a

similar way in both cases, since both have to upload the VMI again.



(a) Cost percentage: 1GB. (b) Cost percentage: 2GB. (c) Cost percentage: 5GB.

Figure 3: Simulation: 2 VMs, 2 cores, different VMI sizes.



(a) Cost: 5 VMs 10 cores. (b) Cost: 5 VMs 15 cores

Figure 4: Combination of instance types + get VMI.

Finally, we have to notice that the storage cost (linked to right axis) remains equal in each case as expected, since the VMI's size is similar in all of them. Moreover, the results show that the deletion policy is more deterministic than the storage policy.

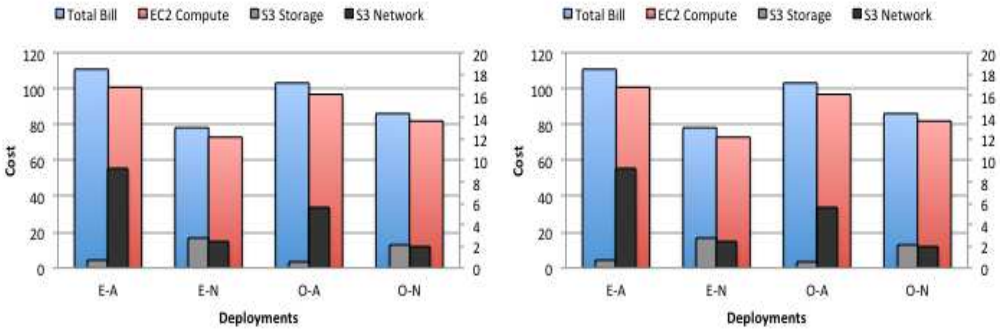
Results from Figures 2(d), 2(e), and 2(f) show the percentage of storage cost over the overall cost. It can be observed how `N` policies have a higher monetary cost on storage than `A` policies. This is obvious as `A` policies delete the VMI when it is no longer used. Moreover, the more VMs we deploy in the same Cloud, the less percentage of storage cost we will have. It can be easily explained as we only need one VMI per Cloud whatever the number of VMs we start in each Cloud.

In Figure 3, we present the results of running experiments with different size of VMIs (1GB, 2GB, and 5GB) to see if our previous result with a VMI size of 0.5GB can be applied to bigger VMI size. It can be observed that the aforementioned best combination of policies (`E-N`) is confirmed as the best

one whatever the combinations of VMs, cores selected, or VMI size are. And obviously, the bigger the VMI is, the higher percentage of storage use we obtain.

The following experiments are requiring more cores per VM. Accordingly, *small* instance type (1 core) is not enough to reach the goal. Hence, the algorithm has to use multiple instance types in these experiments. For instance, we require 10 cores with 5 VMs in Figure 4(a), and 15 cores with 5VMs in Figure 4(b). This fact can lead to use more than one region at once. Accordingly, the broker must upload the VMI to different regions, so the total amount of data uploaded can be doubled or tripled in the worst case, *i.e.* 1 XL (4 cores), 2 L (2 cores), and 2 S (1 core), to achieve 10 cores using 5 VMs, and deploying them in three different regions. It confirms the intuition we described in Section 4: "on-demand" and "always" policies are increasing the startup time of VMs, and also that, even with larger number of VMs, `E-N` is still the best one.

However, we noticed that this kind of experiments have some inconvenients. As the VMIs have to be uploaded in several Clouds, the price could have changed meanwhile the VMs are being started, so it renders unusable the decision taken by the Cloud brokering algorithm.



(a) Cost: 5 VMs 10 cores.

(b) Cost: 5 VMs 15 cores.

Figure 5: Combination of instance types + copy VMI.

To address this issue, we have studied another way of uploading the VMI and we have implemented it into the simulator. It uses the option offered by Amazon to copy data from one S3 region to another. Copying files among regions is quicker than upload the VMI from the broker, so once the first upload has been made, it is possible to copy the VMI to the location indicated by the algorithm.

Therefore, we have studied the impact of the different strategies to handle the transfer of VMI, *i.e.* `Get`, and `Copy`. The results are presented in Figures 5(a) and 5(b). We can observe in both figures that the storage consumption follows the same trends as previously expected. The storage cost is higher for the cases where the `N` strategy is used than the cases where it is the `A` that is used. Furthermore, the data transfer cost is higher for the cases that use the `A` strategy than the ones that use the `N` strategy as more data is transfered between regions.

6 Conclusions and Future Work

In this work we have presented an extended version of our previous Cloud brokering algorithm that takes into account data storage. Although other works exist on Cloud brokering algorithms, none of them take into account data storage and transfer. As a result, we have proposed two strategies for VMI storage, two strategies for VMI deletion, and also we have introduced two different VMI transfer strategies. In the experimental section, we have studied through simulation our modified algorithm under different combinations of VMI storage, deletion, and transfer strategies. Our experimental results highlight the significance of data transfer, deletion and storage policies for multi-Clouds environments. Thanks to the experiments, we can conclude that keeping images in every Cloud results in higher storage costs but lower final bill. Although total cost reduction is quite good, this work is mainly focused on select the best strategies and apply it to the brokering algorithm, instead of obtaining cost improvement percentages. We have also shown that although using a copy mechanism between Clouds can be more expensive, the final bill became lower because of the reduced transfer time.

As future work, we have observed that the algorithm takes too much time when dealing with a higher number of VMs, and the deployment of hundred or thousand of VMs is really interesting in some environments like HPC or for scalability reasons. Therefore, we plan to work on heuristics that do not explore the full set of solutions looking for the optimal one, but try to approximate it by exploring a reduced subset of them. Moreover, we are interested in taking in consideration data transfer costs for brokering mechanisms. Amazon EC2 bills for many different types of data transfer, and this should be taken into account in case, for instance, of tightly coupled VMs.

Acknowledgments

The research leading to these results has received funding from Consejería de Educación of Comunidad de Madrid, Fondo Europeo de Desarrollo Regional, and Fondo Social Europeo through MEDIANET Research Program S2009/TIC-1468; and from Ministerio de Economía y Competitividad of Spain through research grant TIN2012-31518 (ServiceCloud). This work is also partially supported by the french ANR (Agence National de Recherche), project reference ANR 11 INFRA 13 (SONGS).

References

[1] Amazon. Amazon Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2/, June 2013.

[2] R. Buyya, Chee Shin Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, pages 5 –13, September 2008.

[3] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging {IT} platforms: Vi-

sion, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[4] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, January 2011.

[5] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, UKSIM '08, pages 126–131, Washington, DC, USA, 2008. IEEE Computer Society.

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

[7] Frédéric Desprez and Jonathan Rouzaud-Cornabas. SimGrid Cloud Broker: Simulating the Amazon AWS Cloud. Technical Report RR-8380, INRIA, October 2013.

[8] Ana Juan Ferrer, Francisco Hernandez, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, Raul Sirvent, Jordi Guitart, Rosa M. Badia, Karim Djemame, Wolfgang Ziegler, Theo Dimitrakos, Srijith K. Nair, George Kousiouris, Kleopatra Konstanteli, Theodora Varvarigou, Benoit Hudzia, Alexander Kipp, Stefan Wesner, Marcelo Corrales, Nikolaus Forgo, Tabassum Sharif, and Craig Sheridan. OPTIMIS: A Holistic Approach to Cloud Service Provisioning. *Future Generation Computer Systems*, 28(1):66 – 77, 2012.

[9] S.K. Garg and R. Buyya. Networkcloudsim: Modelling parallel applications in cloud simulations. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 105–113, 2011.

[10] Narendra Jussien, Guillaume Rochart, and Xavier Lorca. Choco: an Open Source Java Constraint Programming Library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, Paris, France, France, 2008.

[11] J.L. Lucas-Simarro, R. Moreno-Vozmediano, R.S. Montero, and I.M. Llorente. Scheduling strategies for optimal service deployment across multiple clouds. *Future Generation Computer Systems*, in press, 2012.

[12] J.L. Lucas-Simarro, R. Moreno-Vozmediano, R.S. Montero, and I.M. Llorente. Cost optimization of virtual infrastructures in dynamic multicloud scenarios. *Concurrency and Computation: Practice & Experience*, 2013.

[13] Alberto Nunez, JoseL. Vazquez-Poletti, AgustinC. Caminero, GabrielG. Castané, Jesus Carretero, and IgnacioM. Llorente. icancloud: A flexible

and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.

[14] Martin Quinson, Cristian Rosa, and Christophe Thiery. Parallel simulation of peer-to-peer systems. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, CCGRID '12, pages 668–675, Washington, DC, USA, 2012. IEEE Computer Society.

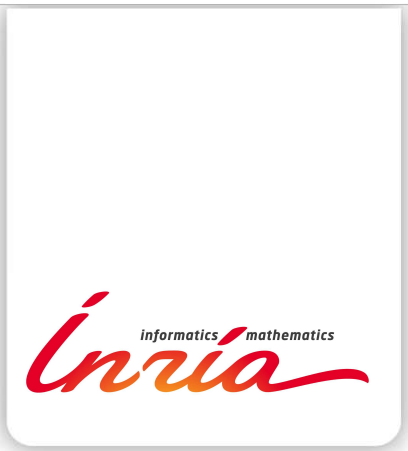
[15] Johan Tordsson, Rubén S. Montero, Rafael Moreno-Vozmediano, and Ignacio M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2):358 – 367, 2012.

[16] Franco Travostino, Paul Daspit, Leon Gommans, Chetan Jog, Ceest de Laa, Joe Mambretti, Inder Monga, Bas van Oudenaarde, Satish Raghunath, and Phil Yonghui Wang. Seamless live migration of virtual machines over the man/wan. *Future Generation Computer Systems*, 22(8):901 – 907, 2006.

[17] Pedro Velho and Arnaud Legrand. Accuracy Study and Improvement of Network Simulation in the SimGrid Framework. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 13:1–13:10, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[18] B. Wickremasinghe, R.N. Calheiros, and R. Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 446–452, 2010.

[19] Sangho Yi, Artur Andrzejak, and Derrick Kondo. Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *IEEE Trans. Serv. Comput.*, 5(4):512–524, January 2012.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

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

ISSN 0249-6399