Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers

Johan Tordsson[a,*], Rubén S. Montero[b], Rafael Moreno-Vozmediano[b], Ignacio M. Llorente[b]

[a] *Dept. of Computing Science and HPC2N, Umeå University, SE-901 87, Umeå, Sweden, www.cloudresearch.se*
[b] *Dept. de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, 28040 - Madrid, Spain, www.dsa-research.org*

Abstract

The last few years we have witnessed the proliferation of a heterogeneous ecosystem of cloud providers, each one with a different infrastructure offer and pricing policy. We explore this heterogeneity in a novel cloud brokering approach that optimizes placement of virtual infrastructures across multiple clouds and also abstracts the deployment and management of infrastructure components in these clouds. The feasibility of our approach is evaluated in a high throughput computing cluster case study. Experimental results confirm that multi-cloud deployment provides better performance and lower costs compared to usage of a single cloud only.

Keywords: Cloud computing, Infrastructure as a Service (IaaS), Scheduling, Interoperability

1. Introduction

As the cloud computing market grows and the number of IaaS providers increases, the market complexity is also increased as users have to deal with many different Virtual Machine (VM) types, pricing schemes, and cloud inter-

*Corresponding author. Telephone: +46 90 786 71 76, Fax: +46 90 786 61 25.
Email addresses: `tordsson@cs.umu.se` (Johan Tordsson), `rubensm@dacya.ucm.es` (Rubén S. Montero), `rmoreno@dacya.ucm.es` (Rafael Moreno-Vozmediano), `llorente@dacya.ucm.es` (Ignacio M. Llorente)

faces. In this context, the use of efficient cloud brokering mechanisms are essential to transform the heterogeneous cloud market into a commodity-like service. These cloud brokers have a two folded role. First, they provide the scheduling mechanisms required to optimize placement of VMs amongst multiple clouds. Second, they offer a uniform management interface with operations, e.g., to deploy, monitor, and terminate VMs, with independence of the particular cloud provider technology.

The scheduling mechanisms required to optimize selection of virtual resources, which can be independent or belong to a multi-component service, amongst different clouds must take into account requirements such as configuration of individual resources, aggregated service performance, total cost, etc. In addition, the user can specify constraints regarding load balancing, service configuration, etc., e.g. to avoid a given set of resources to be allocated in the same cloud (or in different clouds). The cloud scheduler finds an allocation of virtual resources amongst the different cloud providers (a deployment plan) that optimizes the user criteria and adheres to the placement constraints. It is important to note that we are implicitly considering the possibility of a hybrid deployment, i.e., the resources can be placed in different clouds. This multi-cloud setup can be suitable for deployment of independent virtual resources or for loosely-coupled multi-component services with no or weak communication requirements. In case of tightly-coupled services with strong communication requirements or latency-sensitive ones, the service configuration constraints should be used to guarantee single-cloud deployment.

Notably, another important task of a cloud broker is to provide a uniform management interface to deploy, pause, resume, shutdown, monitor, etc. VMs in any cloud, with independence of the particular provider technology. There is, despite multiple ongoing efforts [1, 2, 3], currently no agreed-upon mechanism to interface with a cloud to perform these actions, but rather, each provider exposes its specific API. Thus, a cloud broker must, in order to interface multiple providers, use a software adapter layer to translate between generic management operations and provider-specific APIs.

In summary, our contributions are the following. We propose an architecture for cloud brokering and multi-cloud VM management. We also describe algorithms for optimized placement of applications in multi-cloud environments. Our placement model incorporates price and performance, as well as constraints in terms of hardware configuration, load balancing, etc. An evaluation against commercial clouds demonstrates that compared to single-cloud deployment, our multi-cloud placement algorithms improve performance, lower costs, or provide a combination thereof.

The rest of this paper is organized as follows. The proposed cloud brokering architecture is described in Section 2. Section 3 outlines the problem statement and the mathematical formulations for the cloud scheduling algorithms. The environment for the experimental evaluation is described in Section 4. Sections 5 and 6 describe the obtained scheduling results and their validation in real-world environments. Related work is discussed in Section 7. Section 8 contains conclusions and future directions for this work, followed by acknowledgments and a list of references.

2. Cloud brokering architecture

Figure 1 outlines the cloud brokering architecture used in this work and also illustrates the three roles in the herein studied cloud brokering scenario: the user, the cloud providers, and the cloud broker. A user of the cloud broker requests a virtual infrastructure using a service description template. This template consists of a set of virtual resources that may include compute, network and storage; an optimization criteria, e.g. the total infrastructure capacity; and a set of constraints, e.g. the maximum number of VMs of a certain type. Section 3 discusses optimization parameters in more detail.

Each cloud provider offers several VM configurations, often referred to as instance types. An instance type is defined in terms of hardware metrics such as main memory, CPU (number of cores and clock frequency), available storage space, and price per hour. From a cloud broker architecture perspective it does not matter whether prices are static or determined dynamically, i.e., on-demand lookup of current prices upon each brokering request. In the evaluation against contemporary cloud providers described in Section 4, flat hourly prices are used as this is the only model offered under comparable terms by all providers. Alternative pricing schemes and their implications for cloud brokering are further discussed in Section 8.

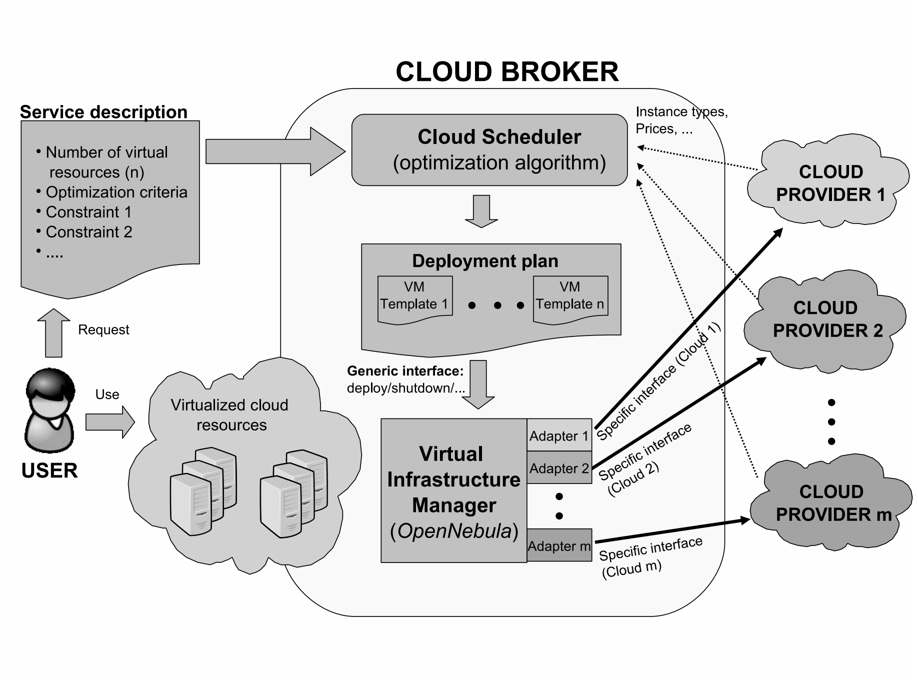

Figure 1: Architecture overview that illustrates the roles in a cloud brokering scenario and outlines the operation of the cloud broker.

The cloud broker performs two main actions: (i) the optimal placement of the virtual resources of a virtual infrastructure across a set of cloud providers; and (ii) management and monitoring of these virtual resources. Based on the infrastructure criteria and constraints provided by the user, the offerings of the available cloud providers, and the used scheduling algorithm, the scheduler component of the cloud broker generates an optimal deployment plan. The deployment plan embodies an explicit implementation of the user's abstract infrastructure request and it contains a list of VM templates. Each template includes the target cloud provider to deploy the VM in as well as attributes specific for the selected provider, e.g., the identifier of the VM image in that cloud.

The cloud scheduling problem can be addressed using either a static or a dynamic approach. The static approach is suitable for situations where the number of required virtual resources is constant (for example, a fixed-size service, a virtual classroom, etc.), and the cloud provider conditions (resource prices, resource availability, etc.) do not change throughout the service life-cycle. In this scenario, the resource selection can be done off-line, once only, and in advance to service deployment. Conversely, the dynamic approach is more suitable for variable size services (e.g., a web server cluster with a fluctuating resource requirements), or in the case of changing cloud provider conditions (variable prices, dynamic resource availability, etc.). In this case, the optimization algorithm should run periodically to adapt the resource selection to the variable service resource requirements and cloud conditions. As current cloud provider conditions are static (prices change very rarely, resource availability is unknown but assumed to be high, etc.) the practical evaluation in this paper is focused on the static approach and a fixed-size service use case, whereas dynamic cloud scheduling mechanisms will be investigated in future work.

To handle the second task of the cloud broker, the Virtual Infrastructure Manager (VIM) provides an abstraction layer on top of the heterogeneous set of clouds, where each cloud has a different interface. This component is responsible for the deployment of each VM in the selected cloud as specified by the VM template, as well as for the management of the VM life-cycle. The VIM caters for user interaction with the virtual infrastructure by making the respective IP addresses of the infrastructure components available to the user once it has deployed all VMs. In this way, the user has a uniform view of the resources and is unaware of their distribution across the clouds. Furthermore, the resources can be accessed using standard applications, e.g., a secure shell or batch queue system.

In this work, the VIM component is based on the OpenNebula virtual infrastructure manager [4]. OpenNebula provides a uniform and generic user interface to deploy, monitor, and control VMs in a pool of distributed physical resources. The OpenNebula core orchestrates three different areas to efficiently control the life-cycle of a VM: virtualization, image management, and networking. OpenNebula is based on a pluggable architecture to allow specific operations in each of the previous areas (e.g. shutdown a VM or clone a disk image) to be performed by specialized adapters [5]. The use of adapters is a well-known technique to achieve this kind of multi-provider interoperation [6, 7]. In addition to adapters to interoperate with virtualization technologies such as XEN [8], KVM [9], and VMware [10], OpenNebula can also interface various cloud providers. In particular, the Amazon Elastic Computing Cloud (EC2) [11] and ElasticHosts (EH) [12] adapters used in the evaluation part of this work convert the general requests made by the virtualization components of the OpenNebula core to manage VMs through the EC2 and EH APIs. Through these cloud adapters, OpenNebula provides a convenient mechanism to access multiple clouds and abstracts, e.g., the different credentials needed to access each cloud. The OpenNebula user and account management mechanisms allow credentials to be configured on a per-user basis, which provides a good fit to the authentication models used by EC2 and EH. Alternative approaches to cloud interoperability as well as cloud standardization efforts are discussed in Section 7.

3. Cloud scheduling algorithms

We here describe scheduling algorithms for applications in a static cloud brokering scenario. In this scenario, the goal is to deploy n VMs, $v_1, \ldots, v_n$, across the m available clouds, $c_1, \ldots, c_m$, to improve criteria such as performance, cost, or load balance. We consider a fixed number (l) of possible hardware configurations of the VMs. These instance types are denoted $IT_1, \ldots, IT_l$. The performance of a given instance type IT_j is denoted $C_j, 1 \leq j \leq l$. Notably, we make no assumptions about the application characteristics or similar in the model but the performance is rather an abstract metric that can be varied depending on the service application. The hourly price for running a VM of instance type $IT_j, 1 \leq j \leq l$ in cloud $c_k, 1 \leq k \leq m$ is denoted P_{jk}. We here consider a 0-1 integer programming formulation where $x_{ijk} = 1$ if v_i is of type IT_j and placed at c_k, and 0 otherwise. Now, the total infrastructure capacity of the deployed VMs is given by

$$TIC = \sum_{j=1}^{l} C_j (\sum_{i=1}^{n} \sum_{k=1}^{m} x_{ijk}). \tag{1}$$

Similarly, the total price of the deployed infrastructure can be expressed as

$$TIP = \sum_{j=1}^{l} \sum_{k=1}^{m} P_{jk} (\sum_{i=1}^{n} x_{ijk}). \tag{2}$$

In the rest of this work, we use TIC as the objective function to maximize, while considering a maximum TIP and various user defined requirements as additional constraints. Users can specify the following types of deployment restriction constraints:

- *VM hardware configuration constraints* are expressed by restricting the instance type. Let min_i and max_i denote the index of the minimum and maximum instance type of v_i. Now, the deployment is restricted to solutions where $x_{i1k} = 0, \ldots, x_{imin_ik} = 0$ and $x_{imax_ik} = 0, \ldots, x_{ilk} = 0$, for $1 \leq i \leq n$ and $1 \leq k \leq m$.

- *Constraints regarding the number of VMs of each instance type* complement the hardware configuration constraints. Users can for this type of constraint specify $ITmin_j$ and $ITmax_j$, the minimum and maximum number of VMs of type j, $1 \leq j \leq l$. Given these limits, the deployment of the VMs must now adhere to $ITmin_j \leq \sum_{i=1}^{n} \sum_{k=1}^{m} x_{ijk} \leq ITmax_j$.

- *Load balancing constraints* are expressed as the minimum (LOC_{min}) and maximum (LOC_{max}) percent of all VMs to be located in each cloud. This constraint is encoded as $LOC_{min} \leq (\sum_{i=1}^{n} \sum_{j=1}^{l} x_{ijk})/n \leq LOC_{max}, 1 \leq k \leq m$ in the integer programming formulation.

Having a fixed number of VMs to deploy enables us to express instance type constraints for each individual VM and hence fine-tune their configuration, which is useful, e.g., for specifying that a particular VM must have more powerful hardware. This benefit comes at the expense of a static infrastructure size in terms of the number of VMs to deploy. Notably, the size of the deployed infrastructure is, as demonstrated in our experiments, still flexible as the number of CPU cores typically differs among the instance types. Another observation is that our multi-cloud scheduling formulation can be used also for the simplified case of deploying a service in a single cloud by setting $m = 1$.

We remark that the formulations in equations (1) and (2) combined with the list of deployment constraints are too relaxed. The large number of potential solutions to the integer programming problem must be restricted to feasible placements only, i.e., where each VM is placed. More specifically, each VM has to be of exactly one instance type and placed in exactly one cloud. This is enforced by adding an additional constraint, $\sum_{j=1}^{l} \sum_{k=1}^{m} x_{ijk} = 1$ for each $i, 1 \leq i \leq n$.

There are multiple modelling languages and solvers that could be used in the cloud scheduler to solve the herein specified optimization problem. Our choice of modelling language is AMPL [13]. This makes the implementation of the cloud scheduling problem straightforward as AMPL has good support for sets and its syntax is close to mathematical notation. AMPL can also be used with a range of backend solvers for various types of optimization problems For the 0-1 integer programming formulations described here, we use the CPLEX [14] solver.

Regarding scalability, it can be observed that our cloud scheduling formulation is a version of the Generalized Assignment Problem (GAP) [15], known to be NP-hard to solve and APX-hard to approximate. The number of potential solutions to the cloud scheduling problem varies with the user-imposed constraints. In particular, load-balancing constraints drastically reduce the solution space. Possible methods to improve scalability include use of state of the art solver software, a topic discussed in-depth by Atamtürk et al. [16]. Further scalability improvements can be achieved at the expense of quality of solution through various approximation techniques. For VM placement formulations similar to our, recent research results demonstrate the feasibility of methods such as column generation [17] and linear rounding [18]. Finally, we remark that heuristics such as greedy formulations [19] can be employed, with the potential to quickly solve very large scale problems.

4. Experimental environment

In order to evaluate both the feasibility of our cloud brokering architecture and the performance of the proposed scheduling algorithms applications, we deploy a set of computational clusters across contemporary cloud providers and evaluate their performance. For this set of experiments, we consider the ElasticHosts [12] and Amazon EC2 [11] cloud providers. The latter offers two separate clouds, one in the USA, one in Europe. These three clouds are henceforth referred to as EH, EC2-US, and EC2-EU, respectively. We use four different VM instance types, their hardware characteristics and prices are listed in Table 1. In this table, the computing capacity row specifies instance type performance according to the Amazon EC2 compute unit metric. This vendor-given performance metric is complemented with a compute-intensive benchmark. The four instance types are from this point on referred to as small, medium, large, and xlarge. EC2-US and EC2-EU both offer the small, large, and xlarge types. In EH, users can customize the hardware setup of the VMs and hence define arbitrary instance types. As EH does not support VMs with 15 GB memory, we consider only the small, medium, and large instance types for this cloud. All prices in Table 1 are for pay-per-use consumption. The various discount schemes offered by contemporary providers are further discussed in Section 8.

The evaluation is performed in two steps. In the first step, we determine the performance of the four instance types with respect to the cluster computing use case. Optimization of the allocation of VMs across clouds and instance types is then investigated for a range of constraints in terms of budget and load balance.

Table 1: Hardware metrics and prices for instance types.

Instance type	small	medium	large	xlarge
CPU (# cores)	1	1	2	4
CPU (Ghz/core)	1	2	2	2
Memory (GB)	1.7	3.5	7.5	15
Disk (GB)	160	300	850	1700
Computing capacity	1	2	4	8
Provider instance type prices ($/h)				
EC2-US	0.1	N/A	0.4	0.8
EC2-EU	0.11	N/A	0.44	0.88
EH	0.158	0.312	0.71	N/A

In the second step, a few selected VM placement plans are deployed and the performance of these infrastructures is analyzed.

The cluster software used is the Sun Grid Engine (SGE) 6.1 [20]. In our setup, the cluster frontend (the sgemaster node) runs in our local infrastructure whereas all worker nodes execute in clouds and are connected to the cluster frontend using OpenVPN. Figure 2 illustrates an example cluster setup. In order to conveniently deploy various SGE cluster configurations, VM images are prepared with NIS, NFS, and OpenVPN preconfigured. When launched, these VM images install the SGE software from a network file system shared with the cluster frontend and dynamically join the SGE cluster as worker nodes. Two images are required for each of EC2-US and EC2-EU, a 32-bit for small instances and a 64-bit for large and xlarge ones. For EH, only 64-bit images are used, but a unique image (called *server* in EH) must be defined for each VM to be executed concurrently. All images use Ubuntu 8.10, OpenVPN 2.0.9, and NIS 2.19. This setup highlights an additional cloud interoperability issue, namely that virtual machine images are specific for each provider. Our approach with dynamic installation of the application software (SGE) addresses this issue, but only partially as the VM images still need to be prepared with fundamental software for network file systems and user management.

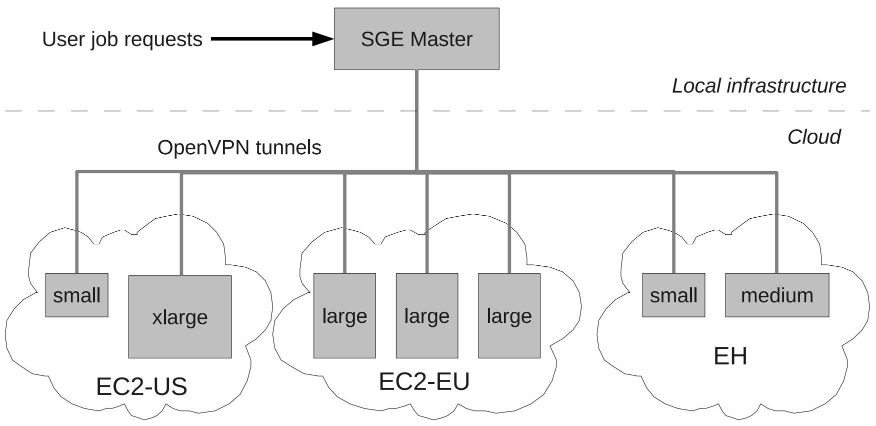

Figure 2: Example SGE cluster configuration with local cluster frontend and cloud worker nodes.

The Embarrassingly Distributed (ED) benchmark from the NAS Grid Benchmarks (NGB) [21, 22] is used to measure the performance of the various instance types and later also of the full-size SGE clusters. This benchmark is representative for the performance of parameter sweep applications, a common problem type within many scientific areas. Notably, it is not our goal to provide an in-depth performance evaluation with respect to a wide range of scientific applications, but rather to demonstrate how the cloud broker, given information about application performance, can optimize the placement of VMs in the clouds. In fact, the related problem of how to assess the performance of scientific applications in loosely-coupled distributed architectures is well studied in the literature. Ostermann et al. [23] investigate the feasibility of using EC2 for scientific applications through a set of benchmarks that evaluate CPU, memory, I/O, etc. Dikaiakos [24] provides a broad summary of topics related to benchmarking of grids, ranging from micro-kernels to evaluation of large-scale infrastructures. Our previous work [25] demonstrates how a grid resource broker can estimate application performance based on benchmark similarity classifications, while avoiding the significant overhead of benchmark (re)execution by extending cluster descriptions in grid information systems with results from standard benchmarks and performance profiles for commonly used applications.

5. Placement optimization results

Table 2 contains the average execution times with standard deviations for three runs of the NGB ED benchmark, size B. In these tests, the number of benchmark jobs were equal to the number of cores of the VM, i.e., two and four benchmark instances were executed concurrently on the larger instance types.

As the CPUs in large and xlarge instances are 2 GHz as compared to 1 GHz in small instances, we expect the benchmark to execute faster on the larger instance types. This can indeed be observed for the EC2-US and EC2-EU clouds, where one benchmark job takes around 700 s to execute in an xlarge VM, whereas a similar job takes between 960 and 980 s in a small VM in these clouds. Notably, for EH, small and medium instances show almost identical performance, which suggests that the vendor-specified hardware metrics are approximative. We also note that for EC2-US and EC2-EU, the benchmark runs faster in large instances than in xlarge ones. This can possible be a consequence of the lower degree of concurrency for the large instance tests (two running benchmark instances compared to four). Regarding the execution time deviations, the performance of EC2-EU and EH seems rather stable, whereas EC2-US shows more volatility, especially for smaller instance types. As the cloud providers offer no SLAs in terms of performance (the Amazon and ElasticHosts SLAs are limited to uptime only) it is not possible to conclude whether these deviations are as expected or not.

Table 2 also illustrates the VM performance (in terms of NGB job throughput) normalized against a small instance in EC2-US. In this part of the table, the fact that the larger instance types execute two and four benchmark instances concurrently is taken into account. When compared to the provider-specified performance (computing capacity in Table 2), we remark that for the NGB ED use case, EC2-US large and xlarge instances run slower than suggested by the computing capacity metric, more specifically at 80% and 70% of the listed capacity. For EC2-EU, the performance of small and large instances is as expected (101% and 95% of stated performance), whereas xlarge instances perform at 71% of specified capacity. Although the performance of the considered instance types is not specified by EH, we compare, as the VM hardware setup is the same as in the other clouds, the measured EH performance with the specification of the instance types in Table 2. We here note that a small instance runs faster than suggested by the EC2-stated performance profile (140% of stated capacity), whereas medium and large instances are slower (70% and 80% of listed capacity). The difference for small may be due to that EC2-EU and EC2-US use 32-bit architectures for small, whereas all other instance types in the three clouds have 64-bit architectures.

The last part of Table 2 shows, for each instance type and cloud, the price-performance ratio in terms on the hourly cost for a unit capacity VM (equivalent to EC2-US small). By showing how cost-efficient the various instance types are, this last part of the table adds some understanding to the results of the scheduling algorithm.

Using AMPL with CPLEX as described in Section 3, we study the TIC objective function (1) for various constraints in terms of TIP (2) and load balance. The placement optimization problem is solved for 16 VMs using both the compute unit metric stated by the cloud providers (see Table 1) and the measured benchmark performance listed in Table 2. For both these cases, the maximum TIP constraint varies from \$1.6 to \$14.6 in steps of \$1. The \$1.6 case corresponds to the cheapest possible solution, i.e., 16 VMs of type small running

Table 2: Benchmarking results of NGB ED size B for each instance type and cloud.

Instance type	small	medium	large	xlarge
Average benchmark execution time (s)				
EC2-US	980 ± 71	N/A	616 ± 61	697 ± 13
EC2-EU	961 ± 12	N/A	513 ± 9	689 ± 13
EH	697 ± 5	694 ± 9	611 ± 18	N/A
Throughput normalized against EC2-US small				
EC2-US	1	N/A	3.2	5.6
EC2-EU	1.01	N/A	3.81	5.67
EH	1.4	1.4	3.2	N/A
Hourly price normalized against throughput				
EC2-US	0.1	N/A	0.125	0.143
EC2-EU	0.109	N/A	0.115	0.155
EH	0.113	0.223	0.222	N/A

in EC2-US, whereas the \$14.6 budget is large enough to allow any placement. In addition to the budget constraints, we study three different load balancing scenarios, i.e., minimum percent of the VMs to be placed in each cloud, with 0%, 10%, and 20% dispersion, respectively.

Figure 3 shows the price-performance results of the optimization problem for various load balancing and budget constraints. In this figure, results are shown for both theoretical, i.e., provider-stated, and measured performance of the considered instance types. The former curves are labelled "(theory)" and the latter ones "(expr)". Figure 3 shows the total cluster capacity in terms of number of completed jobs per minute. This application-specific metric is simply a multiplier of the TIC as one capacity unit (i.e. a small instance in EC2-US) corresponds to 0.061 jobs per minute. Each x value denotes the price of the VM placement, a value typically close to, but never exceeding, the TIP. We remark that the difference between the theoretical and measured performance increases with budget as the difference between provider-stated and benchmarked performance is greater for the more expensive instance types. We also note that as the most expensive and also most powerful instance type (xlarge) is not available in all clouds, the drop in performance caused by higher degrees of load balance increases with the budget. As the performance difference between large and xlarge instances is more significant in theory than in practice, this effect is more prominent in the curves that illustrate theoretical performance. A final observation is that for the theoretical performance values, the total cost of placement never exceeds \$12.8 (corresponding to 16 xlarge VMs in EC2-US á \$0.8/h with a TIC of 128). Contrarily, as the more expensive instance type EC2-EU xlarge (\$0.88/h) performs better than EC2-US xlarge in the benchmarking tests, the practical cluster performance increases also for budgets exceeding \$12.8, with the highest performance achieved for 16 EC2-EU xlarge VMs at a cost of \$14.08/h.

Each point on the curves in Figure 3 denotes the maximum capacity obtainable for a given combination of budget and load balancing constraints. Notably, for each such pair of constraints, many other allocations of VMs over clouds and instance types are possible, but none of these would yield better performance.

As the plotted points dominate all other possible VM placements that adhere to the same constraints, the curves in Figure 3 are the *Pareto frontiers* for the cloud scheduling problem.

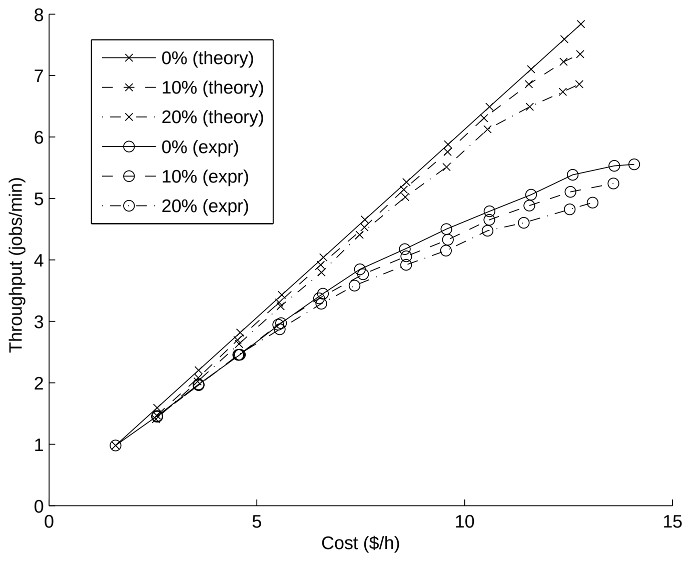

Figure 3: Cost-performance tradeoff for VM placement with varying degree of load balance.

Figures 4, 5, and 6 illustrate the number of VMs of each instance type in each cloud for the optimal solutions. These figures all show the case with benchmark-measured performance. A general pattern observable here is that when the budget is increased by \$1, a few VMs are replaced by ones of more powerful instance types. The price-performance ratios shown in Table 2 give some hints of which instance types and clouds that are the most cost-efficient ones to spend the extra dollar on. Notably, the EH medium instance type is never used, as this instance type has a performance almost identical to that of EH small, but is almost twice as expensive. We also remark that as the least expensive instance type (small) costs more than \$0.1/h in EC2-EU and EH, the cloud scheduling problem for 16 VMs has no solution for 10% and 20% dispersion constraints when the budget is \$1.6. Similarly, as EH does not offer

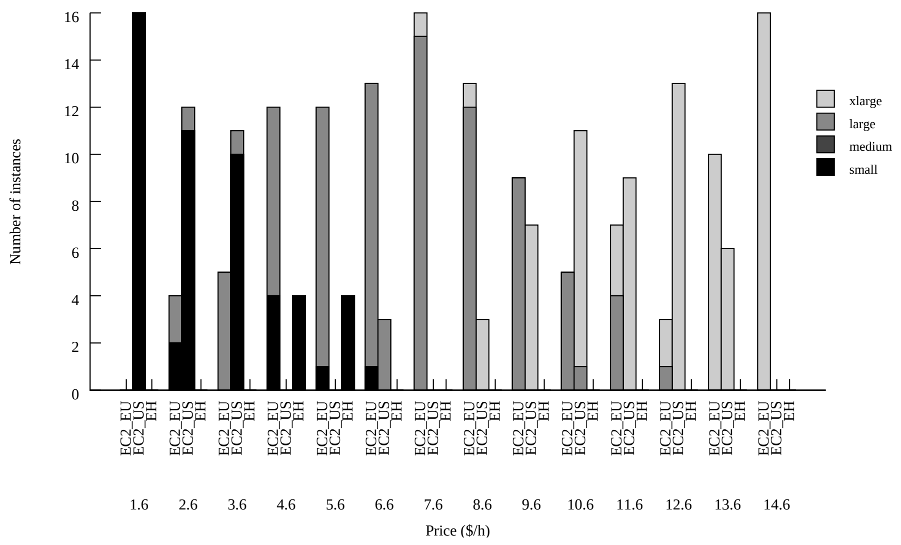

Figure 4: Optimal allocation of VMs for 0% dispersion.

xlarge instances, no improvement in performance is achieved when the budget is increased from \$13.6 to \$14.6 in the 10% and 20% dispersion cases. Notably, also when no dispersion constraints are applied, the optimal solution is in 11 out of 14 cases achieved by placing VMs in more than one cloud.

Although increased dispersion results in lower performance as illustrated by Figure 3, multi-cloud service deployment with load balancing constraints can still be preferable. One benefit of load balancing is the geographical distribution of service components and hence the possibility of bringing these closer to end-users, potentially improving quality of service. Load balancing, when combined with duplication of critical service components, can also provide redundancy and potentially improve fault tolerance. Notably, in such scenarios, careful considerations are needed to ensure redundancy also of the physical infrastructure. The importance of this is illustrated by a recent outage of Amazon EC2, where multiple seemingly redundant infrastructures failed [26].

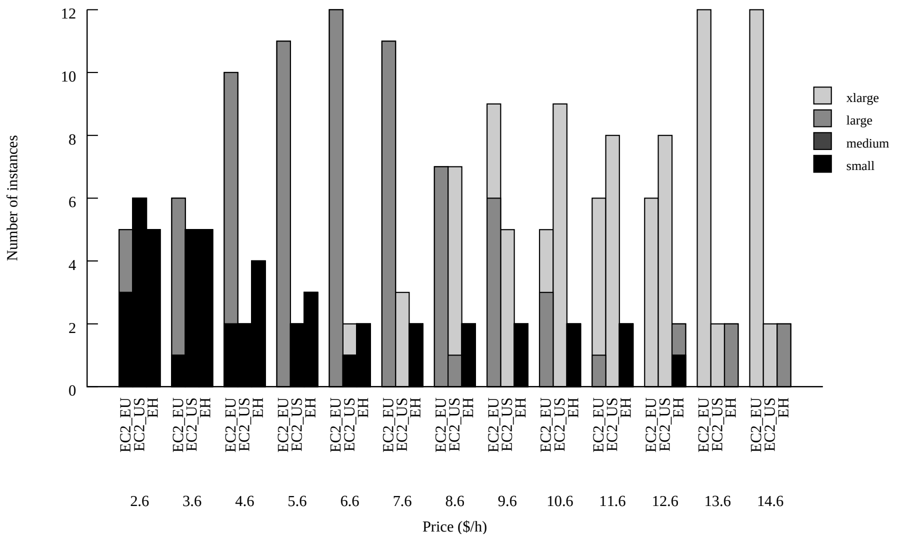

Figure 5: Optimal allocation of VMs for 10% dispersion.

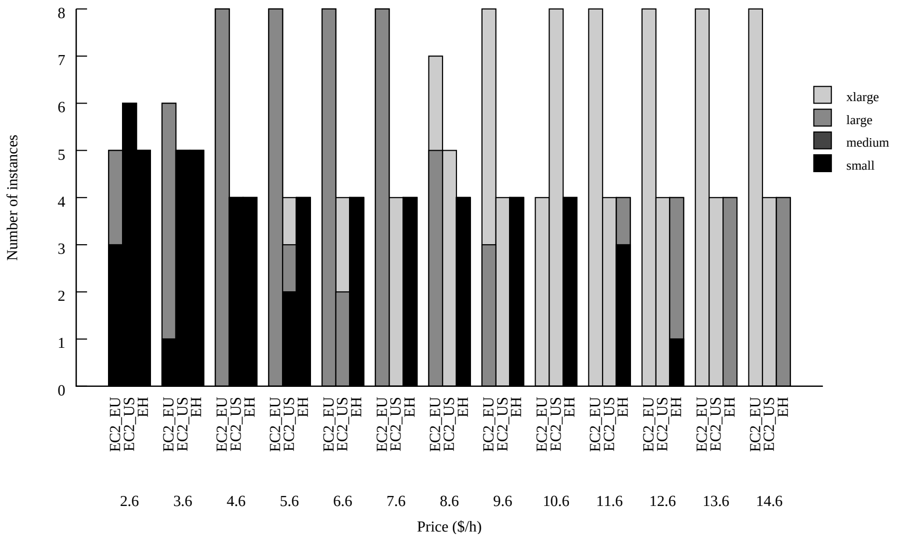

Figure 6: Optimal allocation of VMs for 20% dispersion.

6. Infrastructure benchmarking results

In order to validate the results described in Section 5, a few selected configurations of the SGE cluster are deployed and their performance is evaluated by executing a series of NAS ED benchmark jobs. Here, we focus on budgets \$6.6, \$7.6, and \$8.6, as these (see Figure 3) should give similar performance, while at the same time, include a good mix of different clouds and instance types (see figures 4, 5, and 6). The NAS ED benchmark size B specifies 18 jobs, but this does not saturate the SGE cluster and does hence not allow its asymptotic performance to be studied. Instead, the number of jobs in the tests are eight times the number of cores in the cluster, the latter varying from 31 to 42.

Our previous work [27] demonstrates that the performance of a cluster, in particular for applications like the NGB ED benchmark, can be modelled accurately with a linear approximation:

$$n = r_{\infty} t - n_{1/2}. \tag{3}$$

Here, n is the number of jobs completed by the cluster at time t, r_{∞} is the asymptotic performance, i.e., the maximum job completion rate, and $n_{1/2}$ is the number of jobs required to achieve a job completion rate of $r_{\infty}/2$. From (3) we obtain the cluster throughput (r) in terms of the number of completed jobs (n) per minute:

$$r = n/t = \frac{r_{\infty}}{1 + n_{1/2}/n}. \tag{4}$$

Figures 7, 8, and 9 show the throughput results for the studied cluster configurations. In each of these figures, the x axis shows the completion of individual jobs, and the associated y values (the squares and triangles denoted "Exp.") are the throughputs of the cluster at that point in time. For each set of jobs, the asymptotic throughput as given by (4) is shown as a solid line denoted "Model".

Table 3 summarizes the SGE cluster performance results. In this table, the theoretical performance of a cluster configuration is obtained by adding the performance values of the individual VMs that constitute the cluster (refer to

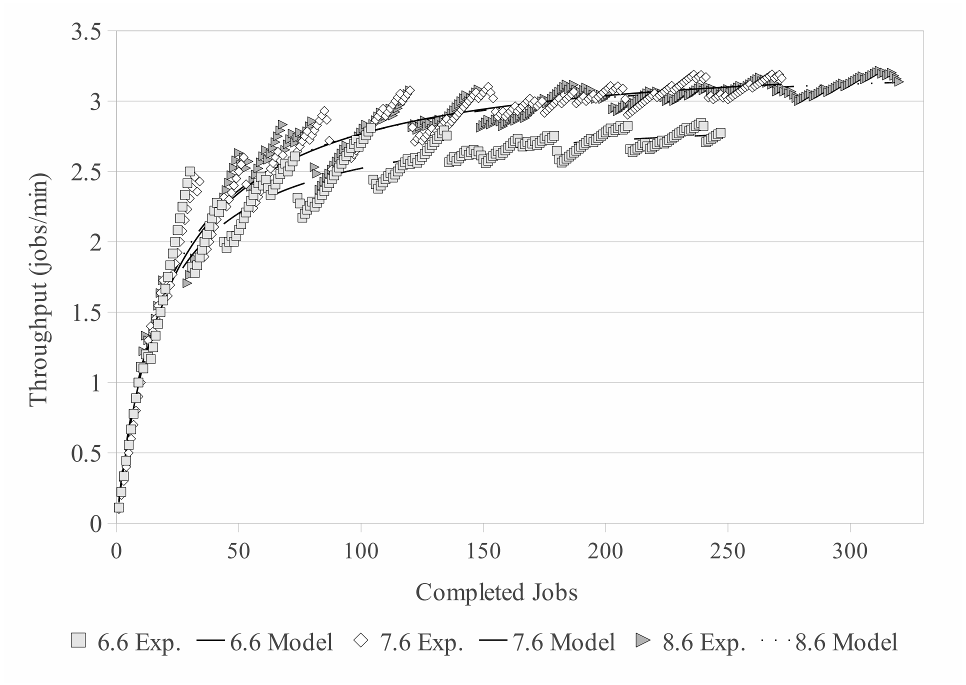

Figure 7: Asymptotic performance of the SGE cluster in the 0% dispersion case.

Table 2 for the individual values). The asymptotic performance (r_∞) is obtained by linear fitting of the benchmark completion times according to (4). These asymptotic values include overhead for SGE job scheduling, network delays, etc. Also included in Table 3 is the measured performance as a ratio of the theoretical one. When comparing the difference between the theoretical peak performance and the measured values, we observe that the practically achievable performance typically is between 80 and 90 percent of the theoretical one. Notably, the measured performance is higher than the theoretical one in two cases. Although this may seem contradictory at first, we remark that the performance values in Table 2 (that form the basis for the theoretical cluster throughput) are the average values from a set of benchmark runs. It could be that the performance of the cluster, overhead included, sometimes is better than the average values in Table 2.

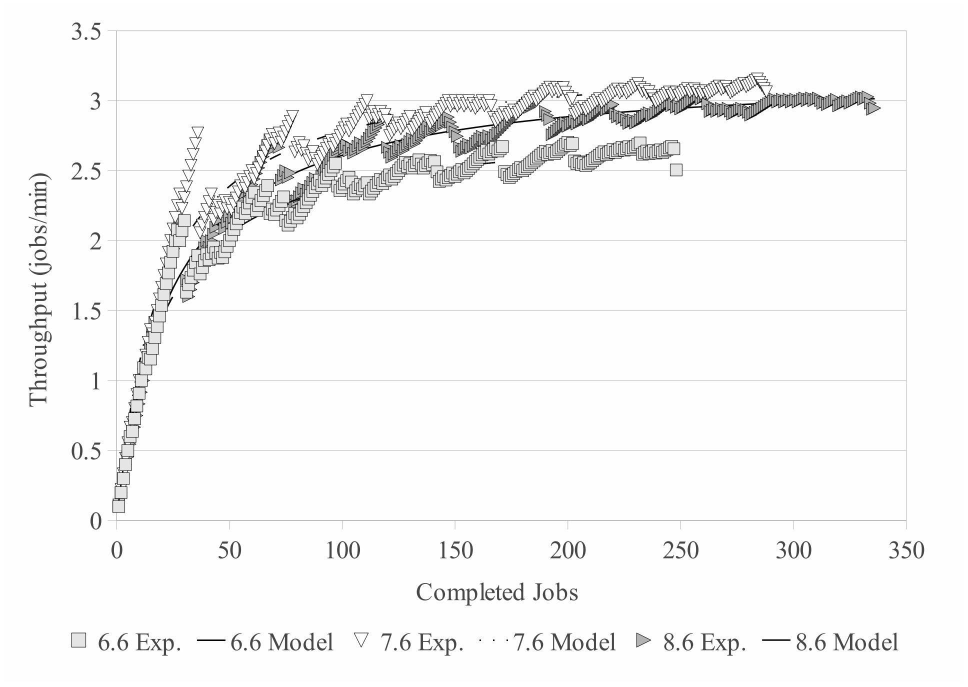

Figure 8: Asymptotic performance of the SGE cluster in the 10% dispersion case.

In order to better understand the differences between practical and theoretical performance, we study the execution times of individual jobs for one of the slowest cases, 0% dispersion \$8.6 budget that achieved 81.9% of the theoretical throughput, and also for one of the fastest cases, 20% dispersion \$8.6 budget that achieved 101.0% of theoretical performance. Figure 10 contains a histogram of the execution times for the 336 jobs (recall that the number of jobs is eight times the number of cores in the cluster, in this case 42) in the 20% dispersion \$8.6 budget case. These are the application wall clock times and excludes overhead introduced by SGE, networks, etc. In this figure, we observe two main groups of execution times, one centered around 510 s and the other around 690 s. The average execution times in Table 2 suggest that these groups correspond well to the instance types used in the experiment. The first group is centered around the average execution time of large instances in

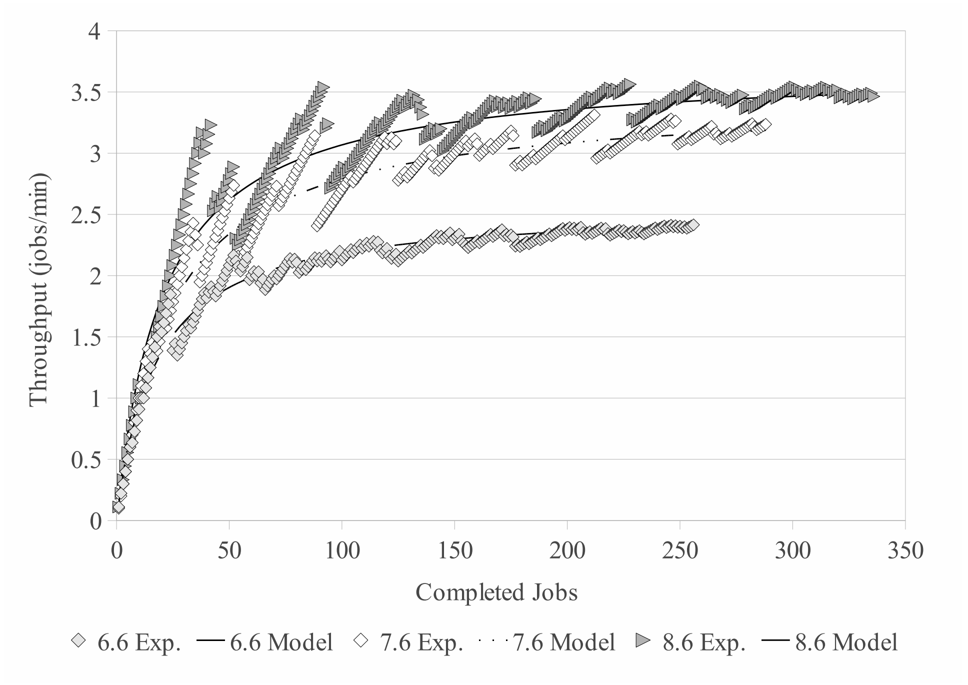

Figure 9: Asymptotic performance of the SGE cluster in the 20% dispersion case.

EC2-EU (513 s), whereas the second one captures the average execution times of EC2-US xlarge instances (697 s), EC2-EU xlarge instances (689 s), and EH small instances (697 s). The number of jobs in each group in Figure 10 fits well with the fact that the five large VMs in EC2-EU constitute around 30% of the total cluster capacity, and hence should execute roughly 30% of the jobs.

Figure 11 shows the execution time histogram of the 320 jobs in the 0% dispersion $8.6 budget case. We here observe a similar execution time pattern with two groups centered around 525 s and 690 s. However, this figure also shows a third prominent group, where the execution times average around 1125 s. This group is not representative for the execution time of any instance type. As the times plotted in Figure 11 are the benchmark execution times, we can exclude a slowdown down due to overhead from OpenVPN, SGE, etc. and conclude that the drop in performance for these jobs is due to high load in

Table 3: Theoretical and measured aggregated performance for selected cluster configurations.

Dispersion	Price ($/h)	Theory (jobs/min)	r_∞ (jobs/min)	Ratio (%)
0%	6.6	3.45	2.9	84.1
	7.6	3.86	3.4	88.8
	8.6	4.03	3.3	81.9
10%	6.6	3.33	2.89	87.6
	7.6	3.61	3.3	91.4
	8.6	3.74	3.23	86.4
20%	6.6	3.18	2.56	80.5
	7.6	3.37	3.45	102.4
	8.6	3.66	3.7	101.0

the cloud(s). By investigating the SGE logs, we notice that the 60 jobs in the slow group are not equally distributed over the SGE worker nodes, but rather focused to three VMs. A pattern observable here is that each of these VMs ran 20 jobs. As these jobs all started and completed within a short span of time in groups of four, we conclude that the VMs were of type xlarge, the only quad core instance type. Of the four xlarge VMs used in the particular experiment (see Figure 4), three were located in EC2-US and one in EC2-EU. However, as the 20% dispersion $8.6 budget experiment used seven xlarge instances from these clouds without suffering from slow jobs, no general pattern can be observed regarding the connection between slowdown and cloud type. We rather suspect variations in cloud load over time to be the cause of the slowdown. Execution time histograms for the other experiments (not included here due to lack of space) indicate a certain degree of cloud overload in seven of the nine cases in Table 3. In the other two cases (20% load balance, budgets $7.6 and $8.6), the practical performance is, SGE and network overhead included, almost identical to the theoretical one.

Similar observations about performance deviations in contemporary cloud

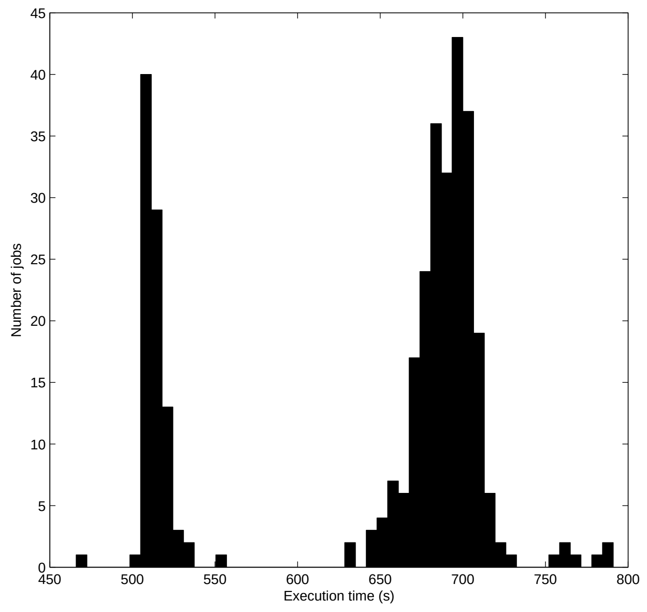

Figure 10: Distribution of benchmark execution times for 20% dispersion and $8.6 budget.

providers are reported in a cloud security study by Ristenpart et al. [28]. Although their primary goal is to determine the likelihood of co-location for VMs, an interesting result in their study is the observation that the time to complete HTTP GET requests varies depending on the number of co-hosted VMs. Other cloud performance deviations are reported in a study by Stantchev [29]. In this study, non-linear performance characteristics are observed for web service benchmarks running on various numbers of EC2 instances.

7. Related work

Resource brokering in federated environments has been thoroughly studied in the context of grid computing, e.g., by Leal et al. [30] who present a decentralized model for scheduling independent tasks in partnering grids. This work is later extended to consider more complex scenarios, where a given or-

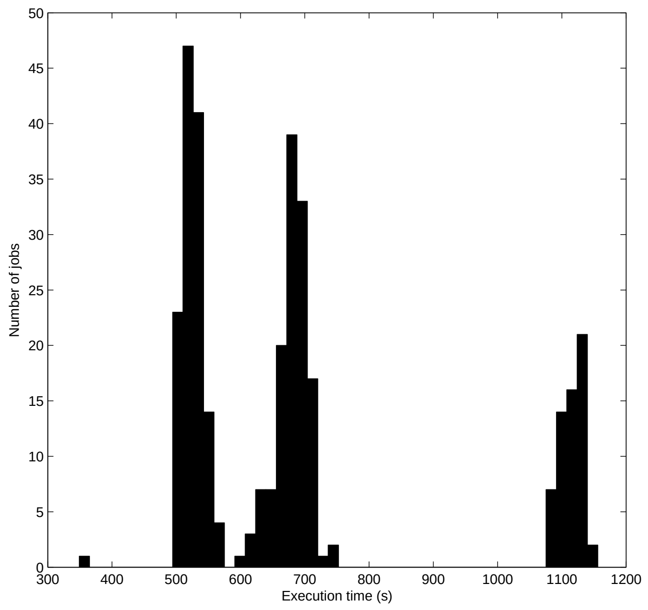

Figure 11: Distribution of benchmark execution times for 0% dispersion and \$8.6 budget.

ganization interfaces with several grid infrastructures of different types [31]. The scheduling policies used in these contributions are mainly guided by performance criteria, trying to optimize throughput, makespan, or network bandwidth usage. Another interesting scheduling issue in federated grid environments is co-allocation, i.e., coordinated advance reservation, of resources that belong to different grids [6]. Resource provisioning policies for multi-grid environments are also studied, e.g., by Assunçao et al. [32]. They analyze provisioning policies that incorporate the cost of resources and demonstrate how a grid can redirect requests to others grids during periods of peak load through a cost-aware load sharing algorithm. Although the use of cost-aware multi-provider scheduling has been explored extensively for Grid environments [33], anomalies, e.g., problems due to the use of artificial currencies [34], suggest that such techniques are more promising for clouds, where resource consumption is based on real financial compensation.

Related research where mathematical programming techniques are used for multi-cloud scenarios include work by Chaisiri et al. who use stochastic integer programming to both minimize the cost of renting virtual machines from public cloud providers and handle uncertainty in future prices and resource demands [35]. Van den Bossche et al. [36] study how to minimize cost of external provisioning in a hybrid clouds. In their studied scenario, partial workloads are outsourced from an internal cloud to public providers. Their work focus on deadline-constrained and non-migratable workloads, where memory, CPU, and networking are incorporated in a binary integer programming problem formulation. Van den Bossche et al. also discuss the tractability of their formulations and provide some experimental insight to scalability. Breitgand et al. [18] propose integer programming formulations for placement of VM workloads within a cloud as well as across multiple clouds collaborating in a federation. They provide a general framework with SLA adherence as the main objective, combined with policies for load balancing and consolidation. Breitgand et al. also provide a 2-approximation for scalability based on linear rounding. Ferreto et al. [19] study various server consolidation schemes for data centers and compare a linear programming formulation for VM placement with greedy heuristics such as first-fit, best-fit, and worst-fit. An experimental evaluation demonstrates that the suggested heuristics perform well in terms of number of physical machines used and VMs migrated, both for academic and industrial workloads.

Standardization of cloud interfaces is an area closely related to the interoperation functionality provided by our cloud broker. Prominent initiatives within cloud infrastructure interoperability include the DMTF Cloud Management Standards that target architectural semantics and implementations for interoperable cloud management between providers [2]. A similarly directed initiative is the OGF Open Cloud Computing Interface (OCCI) working group that defines an API specification for management of cloud computing infrastructures [1]. The cloud standards web page [3] provides an overview of these, and other standardization efforts related to cloud computing. An early vision of cloud interconnection is presented by Buyya et al. [37], who depict a market-oriented architecture for resource allocation within clouds and also share their thoughts on interconnecting clouds to dynamically create global cloud exchanges and markets. Keahey et al. [38] introduce the concept of sky computing, which enables the dynamic provisioning of distributed domains over several clouds. Their paper also discusses the current obstacles for this approach, e.g., image compatibility amongst providers, lack of standards at API level, and shortage of trusted networking environments.

The RESERVOIR [39] initiative combines concepts from grid computing, virtualization, and business service management into an architecture for federated cloud computing. In RESERVOIR, services that constitute multiple VMs are transparently provisioned and managed across clouds, in an on-demand basis, with reduction of costs and preservation of quality of service as management objectives [40]. A similar approach to cloud federation is suggested by Vecchiola et al. [41], who study how the usage of multiple clouds can be used to reduce completion times for deadline-driven tasks. Commercial initiatives in cloud federation include RightScale [42] that has broadened its cloud management platform to support emerging clouds from new vendors, including FlexiScale and GoGrid, while continuing its support for Amazon EC2.

8. Conclusions and future work

This paper proposes a novel approach for cloud brokering in terms of a general architecture for multi-cloud resource provisioning. The main tasks of a cloud broker are to optimize placement of VMs across multiple cloud providers according to user-specified criteria and to manage the resulting infrastructure. By performing this second task, the cloud broker provides an interoperability layer on top of the various cloud provider interfaces. Another contribution is the design of scheduling algorithms for cross-site deployment of applications. The algorithms are based on integer programming formulations and enable price-performance placement tradeoffs. Users can steer the VM allocation by specifying maximum budget and minimum performance, as well as constraints with respect to load balance, hardware configuration of individual VMs, etc.

The cloud brokering architecture and scheduling algorithms are evaluated through a case study where a set of high throughput computing clusters are deployed across contemporary cloud providers. The most important finding in our evaluation is that deployment in more than one cloud improves performance. A multi-cloud placement results in better performance for 11 of 14 budgets (see Figure 4), even though no load balancing constraints are applied. We also note that it is possible to achieve load balancing with 20% of VMs in each cloud (useful, e.g., for bringing services closer to end users) with only a minor decrease in performance compared to an unconstrained placement solution. In some cases, the obtained cluster performance is very close to the theoretical one (see e.g., Figure 10), whereas it sometimes is around 20% slower due to temporary performance declines in the various clouds. An important lesson learned from this is that a cloud scheduling algorithm should, in addition to considering price and performance parameters, also try to model the deviations in cloud provider performance over time.

One difficulty that cloud adopters who interface multiple cloud providers face is the differences in pricing schemes amongst contemporary providers. One issue here is metering, i.e., what hardware metrics billing is based on. Amazon offers fixed size VMs with predefined prices. Additional costs are added for data storage and network transfers. ElasticHosts allows users to customize hardware settings and adjusts the price accordingly. In GoGrid [43], memory, along with network transfers and storage are the main billing metrics. Here, VMs are of fixed sizes similar to those in Amazon. The other main aspect is the billing interval. Amazon offers hourly (pay-per-use) prices, discounts for one and three year subscriptions, as well as spot prices. ElasticHosts uses hourly rates, but offers discounts to users who sign up for at least a month at a time. In ElasticHosts, users also need funds for at least two weeks of continuous use to be able to launch a VM. GoGrid also offers discounts in a prepaid scheme with monthly charges. Although longer term subscriptions can reduce the cost considerately for users with high consumption, selection of a suitable subscription period is a non-trivial task. Development in computer hardware, both in terms of performance improvements and reductions in power consumption (green computing), combined with the economies of scale achieved by large data centers, should with time result in a significant decrease in the hourly prices of VMs. A partial answer to the subscription length problem is given by Walker [44], who models the buy vs lease tradeoff for a large-scale compute cluster. The work by Walker is however limited to fixed pricing schemes.

Future directions for our work include investigation of mechanisms for dynamic cloud scheduling use cases, where the number of VMs change dynamically. A web server is an example application that is suitable for such a scenario as its load can vary substantially over time. One interesting aspect to consider for multi-tier services such as common web applications is affinity and anti-affinity, i.e., how to ensure that certain VMs are deployed in the same (or different) cloud or even physical host [18]. Another aspect to study in web application scenarios is network bandwidth usage and hence also geographical location constraints for the deployed VMs. To ensure end-to-end performance towards users, management control over the network is required, something which is complex to achieve in today's Internet unless dedicated links are used. Another topic of future research interest is dynamic pricing schemes such as Amazon spot prices, which are likely to increase in popularity among provides as these try to maximize the utilization of their infrastructures by offering discounts during periods of low load. Cost minimization in such an environment can likely be achieved by using a *follow the moon* strategy, i.e., placing VMs in clouds that have low load due to off-business hours. Finally, we envision the results of this work to be incorporated in the RESERVOIR architecture [39]. This would allow studies of placement optimization with application-specific performance metrics, research on scheduling based on price-performance tradeoffs governed by SLAs, and investigation of dynamic scenarios where VMs can be migrated between clouds and hence periodically rescheduled.

Acknowledgments

We thank Erik Elmroth, Javier Fontán, Raúl Sampedro, and Constantino Vázquez for their contributions to this work. Financial support is provided by MICINN contract TIN2009-07146 and the Swedish Research Council (VR) contract 621-2005-3667, as well by the European Community's Seventh Framework Programme under grant agreement no. 215605 (RESERVOIR).

References

[1] OGF OCCI - Open Cloud Computing Interface, 2011. Http://www.occi-wg.org, visited June 2011.

[2] DMTF cloud management standards, 2011. Http://www.dmtf.org/standards/cloud, visited June 2011.

[3] Cloudstandards, 2011. Http://www.cloud-standards.org/, visited June 2011.

[4] B. Sotomayor, R. Montero, I. Llorente, I. Foster, Virtual infrastructure management in private and hybrid clouds, IEEE Internet Computing 13 (2009) 14–22.

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns. Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[6] E. Elmroth, J. Tordsson., A standards-based grid resource brokering service supporting advance reservations, coallocation and cross-grid interoperability., Concurrency Computat.: Pract. Exper. 21 (2009) 2298–2335.

[7] E. Huedo, R. Montero, I. Llorente, A Modular Meta-Scheduling Architecture for Interfacing with Pre-WS and WS Grid Resource Management Services, Future Generation Computer Systems 23 (2007) 252–261.

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, ACM SIGOPS Operating Systems Review 37 (2003) 164–177.

[9] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, Kvm: the linux virtual machine monitor, in: Linux Symposium, pp. 225–230.

[10] VMware, 2011. Http://www.vmware.com/, visited June 2011.

[11] Amazon Elastic Compute Cloud (Amazon EC2), 2011. Http://aws.amazon.com/ec2/, visited June 2011.

[12] ElasticHosts, 2011. Http://www.elastichosts.com/, visited June 2011.

[13] R. Fourer, D. M. Gay, B. W. Kernighan, A modeling language for mathematical programming, Management Science 36 (1990) 519–554.

[14] IBM Corporation, 2010. ILOG CPLEX, http://www.ilog.com/products/cplex/, visited May 2011.

[15] H. Kuhn, The hungarian method for the assignment problem, Naval research logistics quarterly 2 (1955) 83–97.

[16] A. Atamtürk, M. W. Savelsbergh, Integer-programming software systems, Annals of operations research 140 (2005) 67–124.

[17] D. Breitgand, A. Epstein, SLA-aware placement of multi-virtual machine elastic services in compute clouds, in: IFIP/IEEE International Symposium on Integrated Network Management (IM'11).

[18] D. Breitgand, A. Marashini, J. Tordsson, Policy-Driven Service Placement Optimization in Federated Clouds, Technical Report, IBM Haifa Labs, 2011.

[19] T. Ferreto, M. Netto, R. Calheiros, C. D. Rose, Server consolidation with migration control for virtualized data centers, Future Generation Computer Systems (2011).

[20] Oracle, 2010. Sun Grid Engine, http://www.sun.com/software/sge/, visited May 2011.

[21] R. V. der Wijngaart, M. Frumkin, NAS Grid Benchmarks Version 1.0, Technical Report, NASA Advanced Supercomputing (NAS) Division, 2002.

[22] M. Frumkin, R. V. der Wijngaart, NAS Grid Benchmarks: a tool for grid space exploration, Journal of Cluster Computing 5 (2002) 247–255.

[23] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, A performance analysis of EC2 cloud computing services for scientific computing, in: Cloudcomp 2009, pp. 115–131.

[24] M. Dikaiakos, Grid benchmarking: vision, challenges, and current status, Concurrency Computat.: Pract. Exper. 19 (2007) 89–105.

[25] E. Elmroth, J. Tordsson, Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions, Future Generation Computer Systems 24 (2008) 585–593.

[26] Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region, 2011. Http://aws.amazon.com/message/65648/, visited June 2011.

[27] R. Montero, E. Huedo, I. Llorente, Benchmarking of high throughput computing applications on grids, Parallel Computing 32 (2006) 267–279.

[28] T. Ristenpart, E. Tromer, H. Shacham, S. 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, pp. 199–212.

[29] V. Stantchev, Performance evaluation of cloud computing offerings, in: Third International Conference on Advanced Engineering Computing and Applications in Sciences, pp. 187–192.

[30] K. Leal, E. Huedo, I. M. Llorente, A decentralized model for scheduling independent tasks in federated grids, Future Generation Computing Systems 25 (2009) 840–852.

[31] K. Leal, E. Huedo, I. M. Llorente, Performance based scheduling strategies for HTC applications in complex federated grids, Concurrency Computat.: Pract. Exper. 22 (2010) 1416–1432.

[32] M. Assunçao, R. Buyya, Performance analysis of allocation policies for intergrid resource provisioning, Information and Software Technology Journal 51 (2009) 42–55.

[33] D. Abramson, R. Buyya, J. Giddy, A computational economy for grid computing and its implementation in the nimrod-g resource broker, Future Generation Computer Systems 18 (2002) 1061–1074.

[34] J. Nakai, Pricing computing resources: reading between the lines and beyond, Technical Report, NASA Advanced Supercomputing Division, 2001.

[35] S. Chaisiri, B. Lee, D. Niyato, Optimal virtual machine placement across multiple cloud providers, in: Services Computing Conference (APSCC) 2009, pp. 103–110.

[36] R. Van den Bossche, K. Vanmechelen, J. Broeckhove, Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads, in: IEEE 3rd International Conference on Cloud Computing (CLOUD), pp. 228–235.

[37] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, Future Generation Computer Systems 25 (2009) 599–616.

[38] K. Keahey, M. Tsugawa, A. Matsunaga, J. Fortes, Sky computing, IEEE Internet Computing 13 (2009) 43–51.

[39] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, F. Galan, The reservoir model and architecture for open federated

cloud computing export, IBM Journal of Research and Development 53 (2009) 1–11.

[40] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, G. Muñoz, H. Tofetti, Reservoir - when one cloud is not enough, IEEE Computer 44 (2011) 44–51.

[41] C. Vecchiola, R. Calheiros, D. Karunamoorthy, R. Buyya, Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka, Future Generation Computer Systems (2011).

[42] RightScale, 2011. Http://www.rightscale.com, visited June 2011.

[43] GoGrid, 2011. Http://www.gogrid.com/, visited May 2011.

[44] E. Walker, The real cost of a CPU hour, IEEE Computer 42 (2009) 35–41.