Cluster Computing manuscript No.
(will be inserted by the editor)

A Cost-Benefit Analysis of Using Cloud Computing to Extend the Capacity of Clusters

Marcos Dias de Assunção · Alexandre di Costanzo · Rajkumar Buyya

Received: 06/11/2009 / Accepted:

Abstract In this paper, we investigate the benefits that organisations can reap by using "Cloud Computing" providers to augment the computing capacity of their local infrastructure. We evaluate the cost of seven scheduling strategies used by an organisation that operates a cluster managed by virtual machine technology and seeks to utilise resources from a remote Infrastructure as a Service (IaaS) provider to reduce the response time of its user requests. Requests for virtual machines are submitted to the organisation's cluster, but additional virtual machines are instantiated in the remote provider and added to the local cluster when there are insufficient resources to serve the users' requests. Naïve scheduling strategies can have a great impact on the amount paid by the organisation for using the remote resources, potentially increasing the overall cost with the use of IaaS. Therefore, in this work we investigate seven scheduling strategies that consider the use of resources from the "Cloud", to understand how these strategies achieve a balance between performance and usage cost, and how much they improve the requests' response times.

This work is partially supported by research grants from the Australian Research Council (ARC) and Australian Department of Innovation, Industry, Science and Research (DIISR). Marcos' PhD research was partially supported by National ICT Australia.

Marcos Dias de Assunção
INRIA RESO/LIP
École Normale Supérieure de Lyon
46, allée d'Italie - 69364 Lyon Cedex 07 - France
E-mail: marcos.dias.de.assuncao@ens-lyon.fr

Alexandre di Costanzo and Rajkumar Buyya
The University of Melbourne
Melbourne, VIC, Australia
E-mail: {adc,raj}@csse.unimelb.edu.au

Keywords Cloud computing · load sharing · job scheduling · backfilling

1 Introduction

Managing and supplying computational resources to user applications is one of the main challenges for the high performance computing community. To manage resources existing solutions rely on a job abstraction for resource control, where users submit their applications as batch jobs to a resource management system responsible for job scheduling and resource allocation. This usage model has served the requirements of a large number of users and the execution of numerous scientific applications. However, this usage model requires the user to know very well the environment on which the application will execute. In addition, users can sometimes require administrative privileges over the resources to customise the execution environment by updating libraries and software required, which is not always possible using the job model.

The maturity and increasing availability of virtual machine technologies has enabled another form of resource control based on the abstraction of containers. A virtual machine can be leased and used as a container for deploying applications [28]. Under this scenario, users lease a number of virtual machines with the operating system of their choice; these virtual machines are further customised to provide the software stack required to execute user applications. This form of resource control has allowed leasing abstractions that enable a number of usage models, including that of batch job scheduling [33].

The creation of customised virtual machine environments atop a physical infrastructure has enabled an-

other model recently known as "Cloud Computing" [2, 38]. Based on the economies of scale and recent Web and network technologies, commercial resource providers, such as Amazon Inc., aim to offer resources to users in a pay-as-you-go manner. These Cloud providers, also known as Infrastructure as a Service (IaaS) providers, allow users to set up and customise execution environments according to their application needs. Previous work has demonstrated how Cloud providers can be used to supply resources to scientific communities. Deelman *et al.* [9] demonstrated the cost of using Cloud providers to supply the needs for resources of data intensive applications. Palankar *et al.* [27] have shown that Grid computing users can benefit from mixing Cloud and Grid infrastructure by performing costly data operations on the Grid resources while utilising the data availability provided by the Clouds.

In this work, we investigate whether an organisation operating its local cluster can benefit from using Cloud providers to improve the performance of its users' requests. We evaluate seven scheduling strategies suitable for a local cluster that is managed by virtual machine based technology to improve its Service Level Agreements (SLAs) with users. These strategies aim to utilise remote resources from the Cloud to augment the capacity of the local cluster. However, as the use of Cloud resources incurs a cost, the problem is to find the price at which this performance improvement is achieved. We aim to explore the trade-off between performance improvement and cost.

We have implemented a system that relies on virtualisation technology for enabling users to request virtual machines from both the local cluster and the Cloud to run applications. In this work, we evaluate via simulation seven strategies for improving scheduling performance through the use of a Cloud provider. In summary, the contributions of this work are to:

- Describe a system that enables an organisation to augment its computing infrastructure by allocating resources from a Cloud provider.
- Provide various scheduling strategies that aim to minimise the cost of utilising resources from the Cloud provider.
- Evaluate the proposed strategies, considering different performance metrics; namely average weighted response time, job slowdown, number of deadline violations, number of jobs rejected, and the money spent for using the Cloud.

The rest of this paper is organised as follows. In Section 2 we provide the background on virtual machines, Cloud computing, and scheduling. Then, we present the seven scheduling strategies for redirecting requests from the cluster to the Cloud in Section 3. Section 4 describes the system design. Next, Section 5 shows the considered experimental scenario and reports the performance evaluation of the investigated strategies. Related work is discussed in Section 6 whereas conclusions are presented in Section 7.

2 Background and Context

This work considers the case where an organisation manages a local cluster of computers through virtual machine technology to supply its users with resources required by their applications. The scenario, depicted in Figure 1, can also represent a centre that provides computing resources to scientific applications or a commercial organisation that provisions resources to its business applications. The organisation wants to provision resources for its user applications in a way that guarantees acceptable response time.

The resources of the *local cluster* are managed by a Virtual Infrastructure Engine (VIE) such as Open Nebula [14] and Eucalyptus [26]. The VIE can start, pause, resume, and stop Virtual Machines (VMs) on the physical resources offered by the cluster. The scheduling decisions at the cluster are performed by the *Scheduler*, which leases the site's virtual machines to the users. The scheduler also manages the deployment of VMs on a *Cloud Provider* according to provisioning strategies, which are detailed in the next section.

2.1 Virtualisation Technologies

The increasing availability of VM technologies has enabled the creation of customised environments on top of physical infrastructures. The use of VMs in distributed systems brings several benefits such as: (i) server consolidation, allowing workloads of several under-utilised servers to be placed in fewer machines; (ii) the ability to create VMs to run legacy code without interfering in other applications' APIs; (iii) improved security through the creation of sandboxes for running applications with questionable reliability; (iv) dynamic provision of VMs to services, allowing resources to be allocated to applications on the fly; and (v) performance isolation, thus allowing a provider to offer some levels of guarantees and better quality of service to customers' applications.

Existing systems based on virtual machines can manage a cluster of computers by enabling users to create virtual workspaces [21] or virtual clusters [14, 6, 15] atop the actual physical infrastructure. These systems can

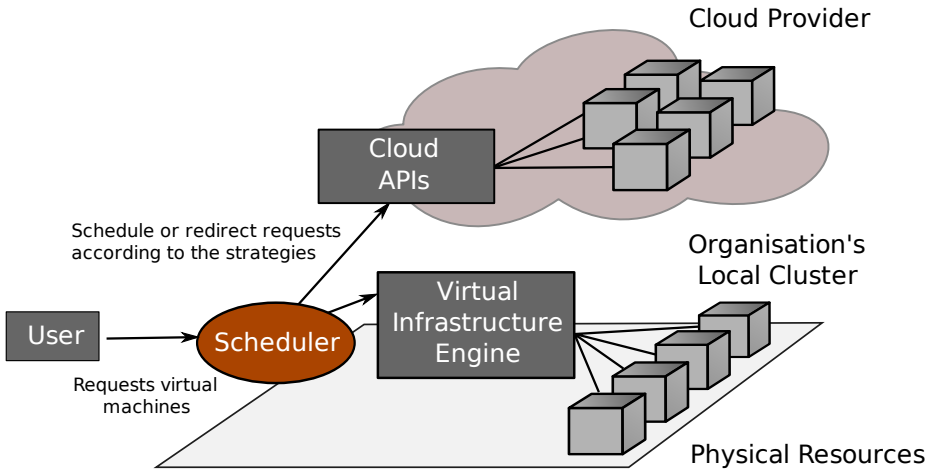

Fig. 1 The resource provisioning scenario.

bind resources to virtual clusters or workspaces according to the demands of user applications. They also provide an interface through which the user can allocate virtual machines and configure them with the operating system and software of choice. These resource managers allow the user to create customised virtual clusters using shares of the physical machines available at the site.

Virtualisation technology minimises some security concerns inherent to the sharing of resources among multiple computing sites. Therefore, we utilise virtualisation software in our system design, described in Section 4, because existing cluster resource managers relying on virtual machines can provide the building blocks, such as availability information, required for the creation of virtual execution environments. The creation of execution environments comprising multiple computing sites is our long-term goal. In addition, relying on virtual machines eases deploying execution environments on multiple computing sites as the user application can have better control over software installed on the resources allocated from the sites without compromising the operation of the hosts' operating systems.

2.2 Infrastructure as a Service

Virtualisation technologies have also facilitated the realisation of new models such as as Cloud Computing or IaaS. The main idea is to supply users with on-demand access to computing or storage resources and charge fees for their usage. In these models, users pay only for the resources they utilise. A key provider of this type of on-demand infrastructure is Amazon Inc. with its Elastic Compute Cloud (EC2) [1]. EC2 allows users to deploy VMs on Amazon's infrastructure, which is composed of several data centres located around the world. To use Amazon's infrastructure, users deploy instances of pre-submitted VM images or upload their own VM images to EC2. The EC2 service utilises the Amazon Simple Storage Service (S3), which aims at providing users with a globally accessible storage system. S3 stores the users' VM images and, as EC2, applies fees based on the size of the data and the storage time.

2.3 Scheduling and Redirection Strategies

The strategies investigated in this work define how the scheduler performs the scheduling of leases and when it borrows resources from the Cloud. The scheduler is divided into two sub-scheduler modules, one managing the scheduling of requests at the local cluster, hereafter also termed the *Site Scheduler*, and another managing the scheduling on the Cloud resources, termed as the *Cloud scheduler*. We term a strategy or algorithm used to schedule the leases as a *scheduling strategy*, and the algorithm that defines when the scheduler borrows resources from the Cloud and which requests are redirected to the Cloud resources as a *redirection strategy*. A combination of scheduling and redirection strategies is a *strategy set*. As discussed later in Section 3, a redirection strategy can be invoked at different times (*e.g.* a job arrival or completion) in different strategy sets.

2.4 Types of User Requests

In addition to the type of virtual machine required and configuration details, a request r is a tuple containing at least $< n, rt, d >$, where n specifies the number of virtual machines required; rt is the ready time, before which the request is not ready for execution; and d is the deadline for request completion. These parameters are sufficient to specify a wide range of virtual machine

requests. As demonstrated latter in Section 5, by making rt larger than the submission time, the user can specify deadline constrained requests that require advance reservation of virtual machines.

The users of the infrastructure run different applications with different computing requirements. Some applications need resources at particular times to meet application deadlines, whereas other applications are not strict about the time when they are given resources to execute as long as they are granted the resources required. The first category of applications is termed as *deadline-constrained* whereas the second category is termed as *best-effort*.

For the purposes of this work, users are to be serviced by virtual machines hosted by an individual computing site; thus the same user request cannot receive resources from both the Cloud provider and the organisation's cluster. Applications that rely heavily on message passing interfaces are generally sensitive to network delays and, despite advances in virtualisation technology [36], may not benefit heavily from using resources from multiple computing sites. In practice, the execution of these applications is generally confined to an individual computer cluster.

We will relax this assumption in future work as applications may present different communication demands. Some applications are composed of tasks that consist of multiple executions of the same program with different input parameters. These applications are often called bag-of-tasks and the tasks generally do not require communication between them; which makes these applications good candidates for utilising resources from multiple sites.

3 Evaluated Strategy Sets

As described in Section 2, a strategy set consists of strategies for scheduling requests at the site and the Cloud, and a redirection strategy that specifies which requests are redirected to the Cloud.

As scheduling strategies we use conservative [25], aggressive [22], and selective backfilling [34]. With conservative backfilling, each request is scheduled (*i.e.* it is granted a reservation) when it arrives in the system, and requests are allowed to jump ahead in the queue if they do not delay the execution of other requests. In aggressive backfilling, only the request at the head of the waiting queue – called *the pivot* – is granted a reservation. Other requests are allowed to move ahead in the queue if they do not delay the pivot. Selective backfilling grants reservations to requests that have waited long enough in the queue. Under selective backfilling a request is granted a reservation if its expected slowdown exceeds a threshold. The expected slowdown of a request r is also called *eXpansion Factor* (XFactor) and is given by Equation 1.

$$XFactor = (wait\ time + run\ time)/run\ time \quad (1)$$

In fact, we use the Selective-Differential-Adaptive scheme proposed by Srinivasan *et al.* [34], which lets the XFactor threshold be the average slowdown of previously completed requests.

The following strategy sets are used for scheduling requests that arrive at the organisation's cluster:

Naïve: both local Site and Cloud schedulers use conservative backfilling to schedule the requests. The redirection algorithm is executed at the arrival of each job at the site. If the site scheduler cannot start a request immediately, the redirection algorithm checks whether the request can be started immediately using Cloud resources. If the request can start on the Cloud resources, then it is redirected to the Cloud, otherwise it is placed in the site's waiting queue.

Shortest Queue: jobs at the site's cluster are scheduled in a First-Come-First-Served (FCFS) manner with aggressive backfilling [22]. The redirection algorithm executes as each job arrives or completes, and computes the ratio of virtual machines required by requests currently waiting in the queue to the number of processors available, similar to the work of England and Weissman [12]. If the Cloud's ratio is smaller than the cluster's, the redirection algorithm iterates the list of waiting requests and redirects requests until both ratios are similar.

Weighted Queue: this strategy is an extension of the Shortest Queue strategy. As each job arrives or completes, the scheduler computes the number of virtual machines required by waiting requests on the cluster and how many virtual machines are in execution on the Cloud. The site scheduler then computes the number of VMs that can be started on the Cloud, num_vms, as the minimum between the number of VMs demanded by the site's requests and the Cloud's VM limit, and redirects requests to the Cloud until num_vms is reached.

Selective: the local site uses the selective backfilling scheme described earlier. As each job arrives or completes, the scheduler checks which requests can be started, then starts them. Using the same approach based on queue ratios used in the Shortest Queue strategy, the scheduler then computes the ratios for the cluster and the Cloud. If the ratios are different, the algorithm iterates the list of waiting requests and checks their XFactors. For each waiting request, if the expansion factor exceeds the threshold, the algorithm checks

the potential start time for the request at both the Cloud and the site. The algorithm finally makes a reservation at the place that provides the earliest start time.

We also investigate strategies to schedule deadline constrained requests using resources from the site and the Cloud provider. The additional deadline-aware strategy sets are:

Conservative: both local site and Cloud schedule requests using conservative backfilling. As each request arrives, the scheduler checks if the site can meet the request's deadline. If the deadline cannot be met, the scheduler checks the availability on the Cloud. If the Cloud can meet the request's deadline, then the request is scheduled on the Cloud resources. If the request deadline cannot be met, the scheduler schedules the request on the local site if it provides a better start time than the Cloud. Otherwise, the request is redirected to the Cloud.

Aggressive: both local site and Cloud use aggressive backfilling to schedule requests. Similarly to the work of Singh *et al.* [32], as each request arrives the scheduler builds a tentative schedule for currently waiting requests. Using aggressive backfilling for building the tentative schedule, the scheduler sorts the requests applying an Earliest Deadline First scheme and checks whether the acceptance of the arriving request would break any request deadline. If there are no potential deadline violations, the request is scheduled locally; otherwise, a tentative schedule is built for Cloud resources. If the request does not break deadlines of requests scheduled to use the Cloud, the request is served with resources from the Cloud provider. If the request deadline cannot be met, the scheduler schedules the request using the local site's resources if they provide a better start time than the Cloud. Otherwise the request is served by resources from the Cloud.

Conservative with Reservation Support: both local site and Cloud schedule requests using conservative backfilling with support for advance reservation of resources. As each request arrives, the scheduler checks whether it is a best-effort or reservation request. In the first case, the request is placed in the local site. Otherwise, for an advance reservation request the scheduler first checks if the site can provide the resources during the required time-frame. If there are not resources available during the requested time-frame, the scheduler checks the resource availability on the Cloud. The request is then scheduled on the Cloud if it can provide the resources required; otherwise the reservation request is rejected.

Although as of writting of this paper some Cloud providers do not support advance reservation, that does not impact our system because reservations are managed by an entity (*i.e.* Gateway) that uses the Cloud API to start and stop virtual machines when reservations commence or finish. The assumption of this work is that the Cloud will provide the resources required when reservations are enforced.

4 System Design

In this section, we describe briefly the design of the InterGrid Gateway (IGG), which is analogous to the scheduler and uses a VIE to enforce virtual machine leases granted to users. The names of the components derive from our previous work on the interconnection of computational Grids [8]. A complete description of the implementation and an evaluation of the system is available elsewhere [10].

IGGs can have peering relationships that define under which circumstances they borrow resources from one another (*i.e.* redirection strategies). These peering relationships specify when an IGG seeks to use resources from another IGG and how the IGG evaluates a request for resources from another IGG. The IGG has been implemented in Java, and a layered view of its components is presented in Figure 2.

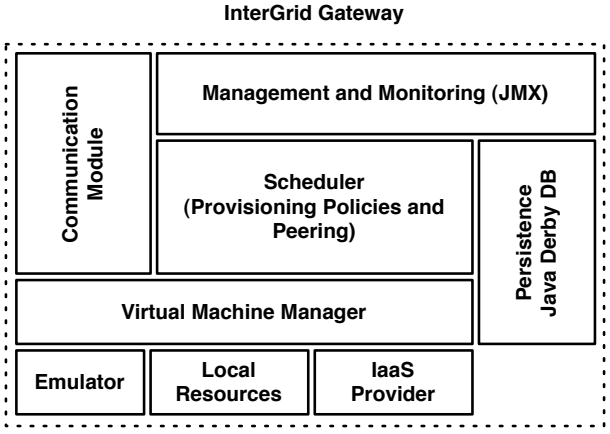

Fig. 2 Main components of the IGG.

The central component of the IGG is the *Scheduler*; in charge of serving users' requests, handling reservations, and managing start and stop of virtual machines when jobs are scheduled. The scheduler maintains the resource availability information and interacts with the Virtual Machine Manager (VM Manager) for creating, starting or stopping virtual machines to fulfil the requirements of the scheduled requests.

The IGG does not share physical resources directly, but relies on virtualisation technology to abstract them. The VM Manager controls the deployment of virtual

machines for the IGG. The types of virtual machines available for the IGG are described as *Virtual Machine Templates*, which are analogous to computers' configurations. A VM template describes a type of VM and contains information such as the number of processors or cores assigned to the VM, the amount of memory, the kernel used to boot the operating system, the disk image, and the price of using a VM of this type over one hour. All available templates are stored in the IGG's repository. At present, users willing to request VMs, need to specify the templates they want to use from the repository. In addition, IGGs need to agree on the templates in order to allow one IGG to borrow VMs from another. In this work, we consider that the Cloud provider has a matching template for each template available at the organisation's cluster.

The VM Manager deploys VMs on physical resources when requested by the IGG. The scheduling strategies that define when and which VMs are started or shut down are implemented as part of the IGG's scheduler. The VM Manager relies on a VIE for deploying and managing VMs; the current implementation uses Open Nebula as a VIE for virtualising a physical cluster infrastructure. In addition, the VM Manager is able to control VMs hosted by a Cloud provider such as Amazon EC2 [1].

The *Communication Module* is responsible for message passing. This module receives messages from other entities and delivers them to the components registered as listeners. Message-passing makes gateways loosely coupled and allows for more failure-tolerant communication protocols.

5 Performance Evaluation

This section describes the scenario considered for performance evaluation, the performance metrics, and experimental results.

5.1 Experimental Scenario

The evaluation of the strategies is performed by using a discrete-event simulator [5]. We use simulation because it enables us to perform repeatable experiments, and the cost incurred by performing experiments on real infrastructure would be prohibitively expensive. To store the information about resources available for running virtual machines, the scheduler uses a data structure based on a red-black tree [7] whose nodes contain the list of resources available at the start or completion of leases. The tree is augmented by a double-linked list connecting the sibling nodes; this list eases the interaction for finding alternative time slots when handling advance reservations or looking for potential start times for requests. This data structure is based on the idea of availability profile used in some implementations of conservative backfilling [25].

For the experiments that do not involve advance reservations, we model the San Diego Super Computer (SDSC) Blue Horizon machine because job traces collected from this supercomputer are publicly available[1] and have been studied previously [23]. The Blue Horizon machine comprises 144 nodes. The experiments with advance reservations model the infrastructure of the Lyon site of Grid'5000[2]. This site has 135 nodes and the requests submitted to it resemble requests for virtual machines; users reserve resources and deploy system images containing the customised operating system and required applications. To model the workload we use traces collected from this site containing one year of request submissions.

The limit of virtual machines that the site can host is the same as the number of nodes. In addition, in this work the maximum number of virtual machines that can be in execution by the Cloud provider at a particular time is the same as the maximum in the local cluster. We plan to relax this assumption in future work.

To compute the cost of using resources from the Cloud provider, we use the amounts charged by Amazon to run basic virtual machines at EC2 (*i.e.* as of writing of this paper the rate was US$0.10 per virtual machine/hour). The experiments consider only the amount charged to run VMs, but in practice Amazon charges for the usage of other resources such as network and storage. Other usage fees are not considered in this work because they depend on the applications' communication and data requirements.

The operating system running on a virtual machine takes from a few seconds to some minutes to boot, but Amazon commences charging users when the VM process starts. The experiments therefore consider that the booting time is already included into the request's duration. In addition, the experiments consider full-hours of utilisation; if a request uses a VM for 30 minutes for example, the cost of one hour is considered.

5.2 Performance Metrics

Some metrics related to requests' response times include the bounded job slowdown (*bound*=10 seconds), hereafter referred only as job slowdown [13] and the

[1] http://www.cs.huji.ac.il/labs/parallel/workload/
[2] http://www.grid5000.fr/

Average Weighted Response Time (AWRT) [16]. The AWRT measures how long on average users wait to have their requests completed. A short AWRT indicates that on average users do not wait long for their requests to complete.

$$AWRT = \frac{\sum_{j \in \tau_k} p_j \cdot m_j \cdot (ct_j - st_j)}{\sum_{j \in \tau_k} p_j \cdot m_j} \quad (2)$$

The AWRT is given by Equation 2, where m_j is the number of virtual machines required by request j, p_j is the execution time of the request, ct_j is the time of completion of the request and st_j is its submission time. The resource consumption ($p_j \cdot m_j$) of each request j is used as the weight.

In order to compute the benefits of using one strategy over another, we also compute the cost ratio between AWRT and the amount spent in running virtual machines on the Cloud. In addition, we measure the number of deadline violations and request rejections when we evaluate scenarios where some requests are deadline constrained. More information about the ratios is provided along with respective experiments.

5.3 Experimental Results

The first experiment evaluates the performance improvement of different strategy sets by running virtual machines on the Cloud provider and the cost of such improvement in each case. This experiment uses a metric termed as *performance cost*. The performance cost of a strategy st is given by Equation 3.

$$perf.\ cost_{st} = \frac{Amount\ spent}{AWRT_{base} - AWRT_{st}} * AWRT_{st} \quad (3)$$

where *Amount spent* is the amount spent running virtual machines on the Cloud provider, $AWRT_{base}$ is the AWRT achieved by a base strategy that schedules requests using only the site's resources and $AWRT_{st}$ is the AWRT reached by the strategy st when Cloud resources are also utilised. This metric aims to quantify the improvement achieved in AWRT and its cost; the smaller the performance improvement cost, the better the strategy performs. In the experiments described in this section, the base strategy is FCFS with aggressive backfilling.

For this experiment, the site's workloads have been generated using Lublin and Feitelson's model [23], here referred to as Lublin99. Lublin99 has been configured to generate two-month-long workloads of type-less requests (*i.e.* no distinction is made between batch and interactive requests); the maximum number of CPUs used by the generated requests is set to the number of nodes in the cluster. This experiment evaluates the performance cost under different types of workloads. In order to generate different workloads, we modify three parameters of Lublin99's model, one at a time. First, we change the mean number of virtual machines required by a request (specified in log_2) to $log_2 m - umed$ where m is the maximum number of virtual machines allowed in system. We vary $umed$ from 1.5 to 3.5. The larger the value of $umed$, the smaller the requests become in terms of numbers of VMs required and consequently result in lighter loads. The second parameter changed in the experiments affects the inter-arrival time of requests at rush hours. The inter-arrival rate of jobs is modified by setting the β of the gamma distribution (hereafter termed $barr$), which we vary from 0.45 to 0.55. As the values for $barr$ increase, the inter-arrival time of requests also increases. The last parameter impacts on the request duration by changing the proportion of the first gamma in the hyper-gamma distribution used to compute the requests runtimes. The proportion p of the first gamma in Lublin99's model is given by $p = pa * nodes + pb$. We vary the paremeter pb from 0.5 to 1.0. The larger the value of pb, the smaller the duration of the requests.

The results of this experiment are shown in Figure 3. Each data point is the average of 5 simulation rounds. Graphs (a), (b) and (c) show the site's utilisation under aggressive backfilling scheduling when the Cloud resources are not used. These graphs illustrate the effect of the parameter changes on the load. Graphs (d), (e) and (f) show the performance cost when we vary: the number of VMs required by a request, the inter-arrival interval and the request's duration, respectively. The higher values obtained by the naïve strategy show that more money is spent to achieve an improvement in AWRT, especially under heavy loads, as shown in graph (d). From graphs (a) and (d), we also observe that the performance cost of using the Cloud is linear with the decrease in number of VMs of requests except for the naïve strategy, which is very expensive for small requests. Under lighter loads, all strategies tend to yield the same ratio of cost and performance. With small inter-arrival periods, all strategies have similar performance, except the naïve strategy. The naïve strategy again provides a high performance cost, as shown in graph (e). With the variation of request arrival time, the experiments show a limit of the performance cost close to US$5,500. The cost increases until this limit

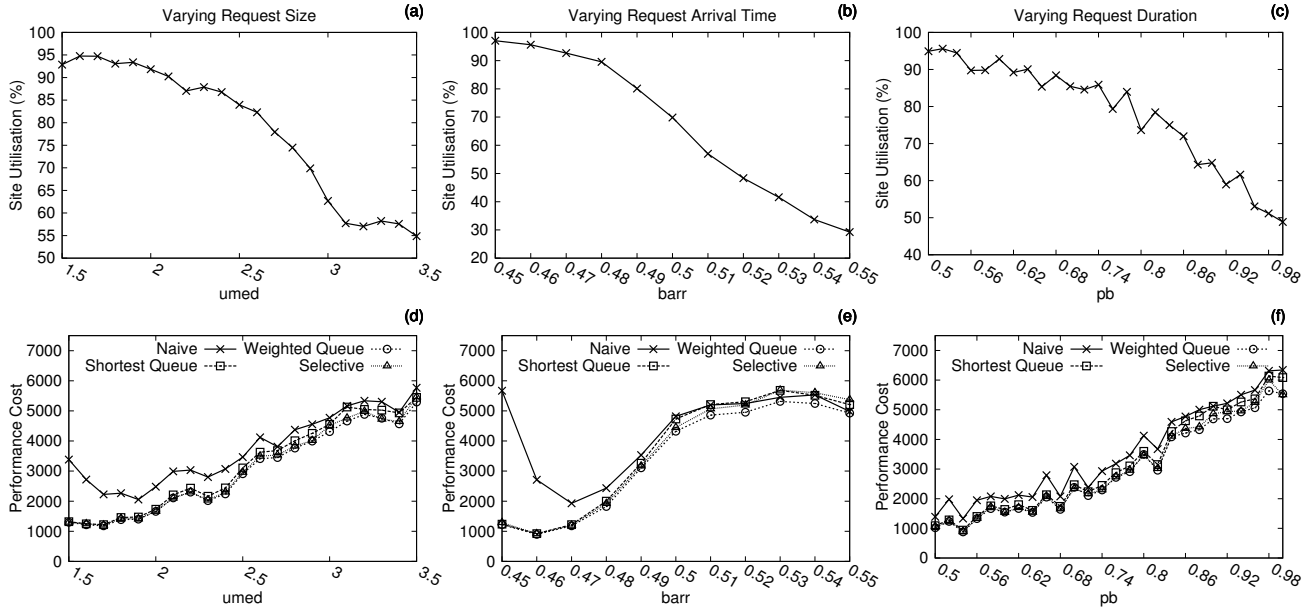

Fig. 3 The top three graphs show the site's utilisation using the base aggressive backfilling strategy without Cloud resources; the bottom three graphs show the performance cost under different workloads. Higher values of *umed* result in requests requiring a larger number of VMs. The larger the value of *barr*, the greater the inter-arrival time of requests at rush hours. The time duration of the requests decrease as the value of *pb* increases. Each data point is the average of 5 simulation rounds.

and then decreases, due to the increase of the request inter-arrival time. More time between requests allows using less resources, which makes it more costly to rely on the Cloud to improve the request response time. For smaller inter-arrival time values, there is an important difference in cost of performance for the naïve strategy in comparison to other strategies. In the last part of the experiment, graphs (c) and (f), all strategies return similar performance cost for the same request duration variation. The performance cost is inversely proportional to the cluster usage.

The second experiment evaluates the site using resources from the Cloud to meet service level agreements with consumers. In this experiment the requests have deadlines and we measure the cost of reducing deadline violations, or requests completing after their deadlines. The cost of reducing deadlines using a strategy st is given by Equation 4.

$$non - violation\ cost_{st} = \frac{Amount\ spent_{st}}{viol_{base} - viol_{st}} \tag{4}$$

where $Amount\ spent_{st}$ is the amount spent with Cloud resources, $viol_{base}$ is the number of violations using a base strategy and $viol_{st}$ is the number of violations under the evaluated strategy. The base policy is aggressive backfilling sorting the jobs for scheduling and backfilling in an Earliest Deadline First manner.

This experiment uses real job traces collected from the SDSC Blue Horizon machine to model the workload of the site's cluster. As the job trace spans a period of two years, we divide it into intervals of two months each. For each experiment, we perform 5 simulation rounds using a different workload for each round. As the deadline information is not available in the trace, we use a Bernoulli distribution to select from the trace the requests that should have deadlines. In this way, a request read from the job trace file has a probability of being deadline constrained. The experiments consider different numbers of deadline constrained requests.

To generate the request deadlines we use a technique described by Islam *et al.* [20], which provides a feasible schedule for the jobs. To obtain the deadlines, we perform the experiments by scheduling requests on the site's cluster without the Cloud using aggressive backfilling. After that, the deadline d_j of a job j is calculated using Equation 5:

$$d_j = \begin{cases} st_j + (ta_j * sf), & if\ [st_j + (ta_j * sf)] < ct_j \\ ct_j, & otherwise \end{cases} \tag{5}$$

where st_j is the request j's submission time, ct_j is its completion time, ta_j if the request's turn around time (*i.e.* the difference between the request's completion and submission times) and sf is a stringency factor that indicates how urgent the deadlines are. If $sf = 1$, then the request's deadline is the completion under the

aggressive backfilling scenario. We evaluate the strategies with different stringency factors (*i.e.* 0.9, 1.3 and 1.7 termed tight, normal and relaxed deadline scenarios respectively).

The results of this experiment are depicted in Figure 4. The top graphs show the amount spent using resources from the Cloud provider to reduce the number of deadline violations. The Conservative and the Aggressive deadline strategies spend smaller amounts than the remaining strategies because they are designed to consider deadlines. Other strategies, except the naïve, sort the requests according to deadlines; however, take into account other performance aspects such as minimising response time when redirecting requests to be scheduled on the Cloud. With a small proportion of deadline constrained requests with tight deadlines, the aggressive strategy had a smaller cost that the conservative strategy. With normal deadlines and a large number of deadline constrained requests, the aggressive strategy spends more than the conservative strategy.

We decided to evaluate the aggressive deadline strategy further in a scenario considering only the site's resources and a case considering the site and the Cloud. If the deadline of a request cannot be met, the request is rejected. This experiment evaluates how much the organisation would need to spend to decrease the number of jobs rejected. The results are summarised in Figure 5.

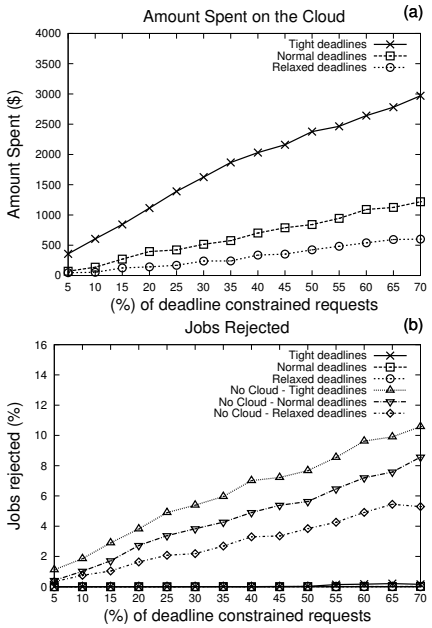

Fig. 5 (a) amount spent using resources from the Cloud provider; (b) the decrease of requests rejected. Each data point is the average of 5 simulation rounds.

Figure 5 (a) shows the amount spent on the Cloud and (b) depicts the percentage of jobs rejected when the Cloud is used and not used. An amount of US$3,000 is spent on the Cloud to keep the number of jobs rejected close to zero under a case where 70% of the requests have deadlines. With normal deadlines, the strategy did not spend more than US$1,500 in any quantity of deadline constrained requests.

Again using traces from the SDSC Blue Horizon, this experiment evaluates the amount of money spent using the Cloud infrastructure under different scheduling strategies, and compares the improvement of the strategies to a scenario where requests were scheduled using only the site's resources with aggressive backfilling. Table 1 summarises the results. All the strategies perform similarly in terms of AWRT improvement. However, the proposed strategy set based on selective backfilling yields a better ratio of slowdown improvement to amount of money spent for using Cloud resources.

The experimental results show that the cost of increasing the performance of application scheduling is higher under a scenario where the site's cluster is underutilised. However, the cost-benefit of using a naïve scheduling strategy can be smaller than using other approaches as a large cost is incurred under scenarios of high system utilisation. In addition, request backfilling and redirection based on the expansion factors (*i.e.* selective backfilling) have shown a good ratio of slowdown improvement to amount of money spent for using Cloud resources.

5.4 Advance Reservations

The experiments discussed in this section measure the cost of handling additional load by using the Cloud to increase the support for reservation of resources. Thereby, we measure the cost of redirecting reservation requests to a Cloud provider and the cost of wasting Cloud resources if the redirected requests fail. As described beforehand, the experiments use a trace collected from a Grid'5000 site containing one year of request submissions. We split the trace into two-month-long periods, and we use a different part for each simulation round. All values reported by the experiments are averages of 5 simulation rounds.

The original request trace contains reservation and best-effort requests. Best-effort requests do not require reservation of resources and their start times are determined by the scheduler; the scheduler places then in the queue and starts their execution at the earliest time when enough resources are available; a best-effort

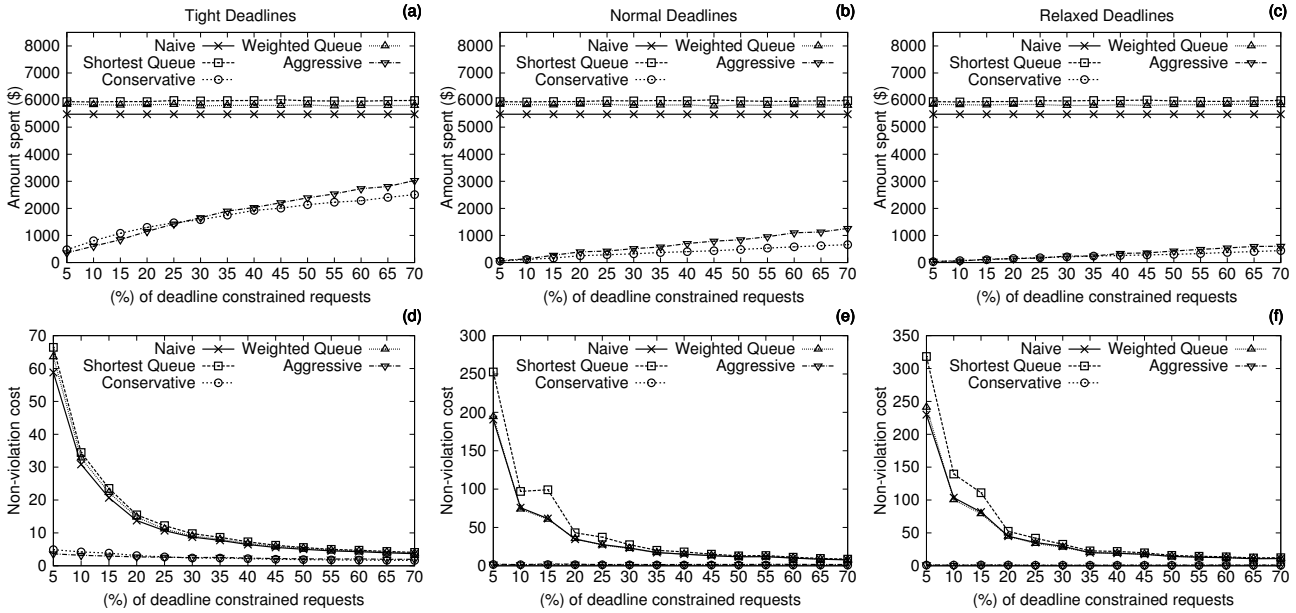

Fig. 4 The top graphs show the amount spent using resources from the Cloud provider; the bottom graphs show the cost of decreasing deadline violations under different numbers of deadline constrained requests and different types of deadlines. Each data point is the average of 5 simulation rounds.

Table 1 Performance of the strategies using workload traces (averages of 5 simulation rounds).

Metric description	Naïve	Shortest Queue	Weighted Queue	Selective
Amount spent with VM instances ($)	5478.54	5927.08	5855.04	4880.16
Number of VM instances/Hours	54785.40	59270.80	58550.40	48801.60
Average weighted response time (improvement)	15036.77	15065.47	15435.11	14632.34
Overall Job slowdown (improvement)	38.29	37.65	38.42	39.70

request can be preempted to make room for a reservation. Whilst we maintain the ratio of reservation and best-effort requests in the experiments, we do not consider preemption of requests; once the schedule of a best-effort request is determined, it is not preempted to give room for a reservation.

To measure the cost of increasing the support for advance reservations, we select randomly from the trace the requests that correspond to the additional load. That is, all the original requests are submitted, along with requests randomly selected from the same trace (*i.e.* additional load). The percentage of requests selected is the additional load and varies from 0 to 100%. Furthermore, the trace contains requests whose executions have failed due to factors such as incorrect configuration of system images and problems with the deployed application. When these requests are redirected to a Cloud provider and the required virtual machines are started, but not fully utilised because the application has failed, the allocation corresponds to a wastage of resources from the organisation's perspective. Although we do not have details about the reasons why these applications have failed in the original execution, we attempt to measure the money spent (or wasted) to allocate resources from the Cloud to serve these applications.

Table 2 summarises the results. The first line shows the amount of money spent using resources from the Cloud under various additional load configurations. The second and the third lines show respectively the number of requests redirected to the Cloud and their corresponding load percentages compared to the overall load generated. The last line shows the amount of money (in US$) spent with requests whose executions have failed. One can observe that all the additional load injected is redirected to the Cloud provider and that the amount spent on the Cloud grows proportionaly to the load redirected. Furthermore, around 60% to 70% of the money spent by using resources from the Cloud were spent on requests whose executions have failed. As discussed beforehand, it is difficult to argue that all these failures reported in the original log have roots on deployment issues; and that is probably not the case. In addition, one can advocate that a commercial provider would offer minimum quality of service and resource availability guarantees that could minimise the number

Table 2 Cost of increasing the support for reservations (averages of 5 simulation rounds).

Additional Load	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Amount spent on the Cloud ($)	77	754	1778	3034	3926	5073	6079	6931	8736	9595	10460
Number of requests redirected	114	1032	1748	2939	4254	5795	7018	9375	11595	14535	16701
Load redirected (%)	0.52	5.20	11.58	19.47	23.31	29.78	32.86	35.68	45.11	48.27	52.15
Amount wasted with failures ($)	49	567	1355	2379	3044	3927	4693	5175	6653	7187	7778

of failures. However, it is important to notice that the results demonstrate that if a commercial Cloud provider is used to extend the capacity of a local infrastructure, deployment of applications and load redirection have to be planned carefully to avoid wastage of resources and consequently waste of money.

6 Related Work

Lease abstractions relying on virtual machine technology have been proposed [33,21,19]. Sotomayor *et al.* [33] explored a lease abstraction to handle the scheduling of a combination of best-effort jobs and advance reservations. Keahey *et al.* [21] demonstrated how to create customised execution environments for a Grid community via Globus Virtual Workspaces. Shirako provides a system of brokers that enable the leasing of various types of resources including virtual machines [19]. In addition, the number of migrations required when the broker and a site scheduler use conflicting policies has been investigated [17]. We evaluate the cost of extending the capacity of an organisation's cluster for improving the response time of user requests.

The applicability of Amazon services for Grid computing has been demonstrated in existing work. Palankar *et al.* [27] evaluated the use of Amazon S3 for Science Grids with data-intensive applications and concluded that Amazon S3 can be used for some of the operations required by data-intensive Grid applications. Although Grid applications can benefit from using Amazon services, such as improving data availability, Palankar *et al.* highlighted that a balance between the benefits of Amazon services and the cost of using Amazon's infrastructure should be taken into account. This balance involves performing expensive operations that generate large amounts of temporary data at the Grid infrastructure. Deelman *et al.* [9] evaluated the cost of using Amazon EC2 and S3 services to serve the resource requirements of a scientific application.

Existing work has shown how to enable virtual clusters that span multiple physical computer clusters [11, 30,31]. Emeneker *et al.* [11] evaluated the overhead of creating virtual clusters using Xen [4] and the Moab scheduler. VioCluster [30] is a system in which a broker responsible for managing a virtual domain (*i.e.* a virtual cluster) can borrow resources from another broker. Brokers have borrowing and lending policies that define when machines are requested from other brokers and when they are returned, respectively. The resources borrowed by one broker from another are used to run User Mode Linux virtual machines.

Systems for virtualising a physical infrastructure are also available. Montero *et al.* [24] investigated the deployment of custom execution environments using Open Nebula. They investigated the overhead of two distinct models for starting virtual machines and adding them to an execution environment. Montero *et al.* [29] also used GridWay to deploy virtual machines on a Globus Grid; jobs are encapsulated as virtual machines. They evaluated several strategies such as using one virtual machine execution per job, pausing the virtual machine between job executions, and reusing the virtual machine for multiple job executions. Montero *et al.* showed that the overhead of starting a virtual machine is small for the application evaluated. We use Open Nebula in the real system implementation of our architecture.

Singh *et al.* [32] proposed an adaptive pricing for advance reservations where the price of a reservation depends on how many jobs it delays. Aggressive backfilling is used to build a tentative schedule and test how many jobs are delayed. We use a similar approach for request admission control in one of our deadline-aware strategies and for deciding on the redirection of requests to the Cloud provider.

Market based resource allocation mechanisms for large-scale distributed systems have been investigated [39]. In this work, we do not explore a market-based mechanism as we rely on utilising resources from a Cloud provider that has cost structures in place. We focus on evaluating the trade-offs between improvement of scheduling user applications and cost of resource utilisation. Specifically, we aim to evaluate the cost of performance improvements.

Several load sharing mechanisms have been investigated in the distributed systems realm. Iosup *et al.* [18] proposed a matchmaking mechanism for enabling resource sharing across computational Grids. Wang and Morris [37] investigated different strategies for load sharing across computers in a local area network. Surana *et*

al. [35] addressed the load balancing in DHT-based P2P networks. Balazinska *et al.* [3] proposed a mechanism for migrating stream processing operators in a federated system. We evaluate the benefits and the cost of adding resources from a Cloud provider to an organisation's infrastructure.

7 Conclusions

This paper evaluated the cost of improving the scheduling performance of virtual machine requests by allocating additional resources from a Cloud computing infrastructure. We considered the case of an organisation that operates its computing infrastructure, but wants to allocate additional resources from a Cloud infrastructure. The experiments evaluated the cost of improving the performance under different strategies for scheduling requests on the organisation's cluster and the Cloud provider. Naïve scheduling strategies can result in a higher cost under heavy load conditions. Experimental results showed that the cost of increasing the performance of application scheduling is higher under a scenario where the site's cluster is under-utilised. In addition, request backfilling and redirection based on the expansion factors (*i.e.* selective backfilling) showed a good ratio of slowdown improvement to the money spent for using Cloud resources.

In future work, we would like to study the performance of different types of applications, such as bag-of-tasks or SPMD running on the local cluster, on the Cloud provider, and both at the same time. In addition, we are currently working on an adaptive strategy that aims to optimise scheduling performance considering the user's budget. For a given budget amount, the scheduler would find the best strategy to fulfil the user's request.

8 Acknowledgments

We would like to thank Sungjin Choi, Suraj Pandey, Carlos Varela, and Marco Netto for their feedback on a preliminary version of this work. We carried out some experiments using traces collected from the Grid'5000 platform, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS, RENATER and other contributing partners (http://www.grid5000.fr).

References

1. Amazon Inc. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2.
2. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of Cloud computing. Technical report UCB/EECS-2009-28, Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, USA, February 2009.
3. M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-based load management in federated distributed systems. In *1st Symposium on Networked Systems Design and Implementation (NSDI)*, pages 197–210, San Francisco, USA, March 2004. USENIX Association.
4. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 164–177, New York, NY, USA, 2003. ACM Press.
5. R. Buyya and M. Murshed. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience (CPE)*, 14(13–15):1175–1220, November-December 2002.
6. J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle. Dynamic virtual clusters in a Grid site manager. In *12th IEEE International Symposium on High Performance Distributed Computing (HPDC 2003)*, page 90, Washington, DC, USA, 2003. IEEE Computer Society.
7. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press / McGraw-Hill, Cambridge, Massachusetts, 2nd edition, 2001.
8. M. D. de Assunção, R. Buyya, and S. Venugopal. InterGrid: A case for internetworking islands of Grids. *Concurrency and Computation: Practice and Experience (CCPE)*, 20(8):997–1024, June 2008.
9. E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the Cloud: The montage example. In *2008 ACM/IEEE Conference on Supercomputing (SC 2008)*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.
10. A. di Costanzo, M. D. de Assunção, and R. Buyya. Harnessing cloud technologies for a virtualized distributed computing infrastructure. *IEEE Internet Computing*, 13(5):24–33, 2009.
11. W. Emeneker, D. Jackson, J. Butikofer, and D. Stanzione. Dynamic virtual clustering with Xen and Moab. In *Frontiers of High Performance Computing and Networking with ISPA 2006*, volume 4331 of *LNCS*, pages 440–451, Berlin/Heidelberg, November 2006. Springer.
12. D. England and J. B. Weissman. Costs and benefits of load sharing in the computational Grid. In *10th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '04)*, volume 3277 of *LNCS*, pages 160–175, New York, USA, 2004. Springer Berling Heidelberg.
13. D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing (IPPS '97)*, pages 1–34, London, UK, 1997. Springer-Verlag.
14. J. Fontán, T. Vázquez, L. Gonzalez, R. S. Montero, and I. M. Llorente. OpenNEbula: The open source virtual machine manager for cluster computing. In *Open Source Grid and Cluster Software Conference – Book of Abstracts*, San Francisco, USA, May 2008.
15. I. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayor, and X. Zhang. Virtual clusters for Grid communities. In *6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2006)*, pages 513–520, Washington, DC, USA, May 2006. IEEE Computer Society.

16. C. Grimme, J. Lepping, and A. Papaspyrou. Prospects of collaboration between compute providers by means of job interchange. In *Job Scheduling Strategies for Parallel Processing*, volume 4942 of *Lecture Notes in Computer Science*, pages 132–151, Berlin / Heidelberg, April 2008. Springer.
17. L. Grit, D. Inwin, A. Yumerefendi, and J. Chase. Virtual machine hosting for networked clusters: Building the foundations for 'autonomic' orchestration. In *1st International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006)*, Tampa, Florida, November 2006.
18. A. Iosup, D. H. J. Epema, T. Tannenbaum, M. Farrellee, and M. Livny. Inter-operating Grids through delegated matchmaking. In *2007 ACM/IEEE Conference on Supercomputing (SC 2007)*, pages 1–12, New York, USA, November 2007. ACM Press.
19. D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing networked resources with brokered leases. In *USENIX Annual Technical Conference*, pages 199–212, Berkeley, USA, June 2006. USENIX Association.
20. M. Islam, P. Balaji, P. Sadayappan, and D. K. Panda. QoPS: A QoS based scheme for parallel job scheduling. In *9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '03)*, volume 2862 of *LNCS*, pages 252–268, Seattle, WA, USA, 2003. Springer.
21. K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the Grids. *Scientific Programming*, 13(4):265–275, 2006.
22. D. A. Lifka. The ANL/IBM SP scheduling system. In *Workshop on Job Scheduling Strategies for Parallel Processing (IPPS'95)*, pages 295–303, London, UK, 1995. Springer-Verlag.
23. U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.
24. R. S. Montero, E. Huedo, and I. M. Llorente. Dynamic deployment of custom execution environments in Grids. In *2nd International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP '08)*, pages 33–38, Valencia, Spain, September/October 2008. IEEE Computer Society.
25. A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.
26. D. Nurmi, R. Wolski, C. Crzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus: a technical report on an elastic utility computing architecture linking your programs to useful systems. Technical Report 2008-10, Department of Computer Science, University of California, Santa Barbara, California, USA, 2008.
27. M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon S3 for science Grids: a viable solution? In *International Workshop on Data-aware Distributed Computing (DADC'08) in conjunction with HPDC 2008*, pages 55–64, New York, NY, USA, 2008. ACM.
28. L. Ramakrishnan, D. Irwin, L. Grit, A. Yumerefendi, A. Iamnitchi, and J. Chase. Toward a doctrine of containment: Grid hosting with adaptive resource control. In *2006 ACM/IEEE Conference on Supercomputing (SC 2006)*, page 101, New York, NY, USA, 2006. ACM Press.
29. A. Rubio-Montero, E. Huedo, R. Montero, and I. Llorente. Management of virtual machines on globus Grids using GridWay. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007)*, pages 1–7, Long Beach, USA, March 2007. IEEE Computer Society.
30. P. Ruth, P. McGachey, and D. Xu. VioCluster: Virtualization for dynamic computational domain. In *IEEE International on Cluster Computing (Cluster 2005)*, pages 1–10, Burlington, USA, September 2005. IEEE.
31. A. Shoykhet, J. Lange, and P. Dinda. Virtuoso: A system for virtual machine marketplaces. Technical Report NWU-CS-04-39, Electrical Engineering and Computer Science Department, Northwestern University, Evanston/Chicago, IL, July 2004.
32. G. Singh, C. Kesselman, and E. Deelman. Adaptive pricing for resource reservations in shared environments. In *8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, pages 74–80, Austin, USA, September 2007. ACM/IEEE.
33. B. Sotomayor, K. Keahey, and I. Foster. Combining batch execution and leasing using virtual machines. In *17th International Symposium on High performance Distributed Computing (HPDC 2008)*, pages 87–96, New York, NY, USA, 2008. ACM.
34. S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Selective reservation strategies for backfill job scheduling. In *8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '02)*, volume 2537 of *LNCS*, pages 55–71, London, UK, 2002. Springer Berlin Heidelberg.
35. S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica. Load balancing in dynamic structured peer-to-peer systems. *Performance Evaluation*, 63(3):217–240, 2006.
36. M. Tatezono, N. Maruyama, and S. Matsuoka. Making wide-area, multi-site MPI feasible using Xen VM. In *Workshop on Frontiers of High Performance Computing and Networking (held with ISPA 2006)*, volume 4331 of *LNCS*, pages 387–396, Berlin/Heidelberg, 2006. Springer.
37. Y.-T. Wang and R. J. T. Morris. Load sharing in distributed systems. *IEEE Transactions on Computers*, C-34(3):204–217, March 1985.
38. A. Weiss. Computing in the Clouds. *netWorker*, 11(4):16–25, December 2007.
39. R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational Grid. *The International Journal of High Performance Computing Applications*, 15(3):258–281, Fall 2001.