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

Enhancing Performance of Failure-prone Clusters by Adaptive Provisioning of Cloud Resources

Bahman Javadi · Parimala Thulasiraman · Rajkumar Buyya

Received: date / Accepted: date

Abstract In this paper, we investigate Cloud computing resource provisioning to extend the computing capacity of local clusters in the presence of failures. We consider three steps in the resource provisioning including resource brokering, dispatch sequences, and scheduling. The proposed brokering strategy is based on the stochastic analysis of routing in distributed parallel queues and takes into account the response time of the Cloud provider and the local cluster while considering computing cost of both sides. Moreover, we propose dispatching with probabilistic and deterministic sequences to redirect requests to the resource providers. We also incorporate checkpointing in some well-known scheduling algorithms to provide fault-tolerant environment. We propose two cost-aware and failure-aware provisioning policies that can be utilized by an organization that operates a cluster managed by virtual machine technology and seeks to use resources from a public Cloud provider. Simulation results demonstrate that the proposed policies improve the response time of users' requests by a factor of 4.10 under a moderate load with a limited cost on a public Cloud.

Bahman Javadi
School of Computing, Engineering and Mathematics
University of Western Sydney
E-mail: b.javadi@uws.edu.au

Parimala Thulasiraman
InterDisciplinary Evolving Algorithmic Sciences (IDEAS) Laboratory
Department of Computer Science
University of Manitoba, Winnipeg, Canada

Rajkumar Buyya
Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia

1 Introduction

Public Cloud platforms provide easy access to an organizations' high-performance computing and storage infrastructures through web services. In this platform, the complexity of managing an IT infrastructure is completely hidden from its users. One particular type of Cloud service, known as Infrastructure-as-a-Service (IaaS) provides raw computing and storage in the form of virtual machines (VMs), which can be customized and configured based on application demands providing massive scalability, high reliability and performance. Although, IaaS can be used as a stand alone service, in this paper, we integrate public Cloud services with that of an organization' local cluster running virtual machine technology. Utilization of public Cloud along with local cluster resources is commonly called *hybrid Cloud* [1], the system used in this paper. Security concerns with the public Clouds is one of the main drivers for hybrid Clouds. High performance computing applications can leverage from such systems to execute data intensive applications for increased performance gain.

In the literature, several works [2–5] have adopted public Cloud platforms for scientific applications. Most of these works, however, only demonstrate performance and monetary cost-benefits for such applications. Recently, Assunção *et al.* [5] proposed scheduling strategies to integrate resources from public Cloud provider and local cluster. In this work, the requests are first instantiated on cluster and in the event more resources are needed to serve user requests, IaaS Cloud provider virtual machines are added to the cluster. This is done to reduce users' response time. Their strategies, however, do not take into consideration the total cost of the hybrid Cloud when making decisions on redirection of requests between local cluster and public Cloud. Furthermore, the authors do not consider the trade-off between cost and performance in case of resource *failures* on local cluster. A failure is defined as an event in which the system fails to operate according to its specifications [6]. In the presence of resource failures, a job could result in premature termination leading to undesirable completion time. In particular, compute bound jobs such as batch programs whose results cannot be used until the jobs are completed in its entirety, suffer due to resource failures.

In this paper, we aim to provide *cost-aware* and *failure-aware* provisioning policies to extend the capacity and enhance the performance of existing local cluster in the presence of resource failures. We consider three steps in the resource provisioning policy in the hybrid Cloud system. The first step is the resource brokering to obtain the proportion of the input workload to be served in each resource provider. For this purpose, we propose a generic analytical model based on stochastic analysis of distributed parallel queues [7]. The second step in the resource provisioning is the dispatch sequence, which is the sequence of submitting requests to providers. We propose two different dispatch methods based on probabilistic and deterministic sequences. The last step is scheduling of requests on resources, which would be done through different well-known scheduling algorithms. Our proposed policies take advan-

tage of non-observable queues, so they do not require any information of the scheduler's queues. We evaluate the proposed policies under realistic workload and failure traces and consider different performance metrics such as average weighted response time and job slowdown.

The rest of this paper is organized as follows. In Section 2, we present the hybrid Cloud system model used in this paper. Related work is described in Section 3. Section 4 presents the generic resource provisioning policy including brokering strategy, dispatch sequences, and scheduling algorithms. The resource provisioning in a hybrid Cloud system with two resource providers is described in Section 5. The performance evaluation of the proposed policies is discussed in Section 6. We also analyze the cost of mapping the proposed policies on a real public Cloud in this section. Conclusions and future work are presented in Section 7.

2 System Model

In this section, we briefly present the hybrid Cloud system model used in this paper. This is based on the InterGrid middleware designed and implemented in the Cloudbus research group[1] [8].

2.1 System Architecture

There are two different types of resource providers in the system as seen in Figure 1: the local cluster running virtual machines (within organization's Site) and public IaaS Cloud provider. A user launches an application on the local cluster and submits a request to the InterGrid gateway (IGG). Each user's request has the following characteristics: type of required virtual machine; number of virtual machines; estimated duration of the request; deadline of the request. When such a request arrives at IGG, the IGG determines which resource provider to use.

An IGG is aware of the peering terms between resource providers and selects suitable one that can provide the required resources for incoming request. The adaptive provisioning policy is a part of the IGG which include the scheduling algorithms of the local resources and the public Cloud as well as brokering strategies to share the incoming workload with the public Cloud provider. An IGG is able to interact with Virtual Infrastructure Engine (VIE) or another IGG to schedule resources on local cluster or IaaS Cloud provider. The VIE manages the resources of the local cluster. The VIE and can start, pause, resume, and stop VMs on the physical resources. IGGs have peering arrangements that allows them to communicate and determine when and how the resources are used between IGGs. In such circumstances, an IGG can interact with another IGG to provision resources from a public Cloud to fulfill the users' requirements.

[1] http://www.Cloudbus.org/

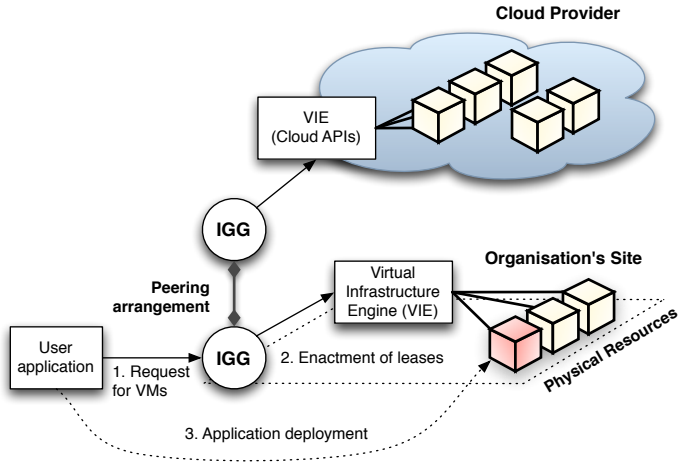


Fig. 1 The system architecture.

A three-step scenario in which an IGG allocates resources from a local cluster in an organization to deploy applications, is indicated in Figure 1. As one can see, the user sends a request for VMs to the IGG. The local IGG tries to obtain resources from the underlying VIE. This is the point where the IGG must make decision about selecting resource provider to supply the user's request, so the resource provisioning policies come to the picture. Once the IGG has allocated the requested VMs, it makes them available and the user will be able to access the VMs and finally deploy the application.

2.2 System Implementation

The IGG has been implemented in Java and a layered view of its components is depicted in Figure 2. The core component of the IGG is the *Scheduler*, which implements provisioning policies and peering with other IGGs. The scheduler maintains the resource availability information as well as creating, starting and stopping the VMs through the Virtual Machine Manager (VMM). VMM implementation is generic, so different VIEs can be connected and make a flexible architecture. Currently, VIE can connect to OpenNebula [9], Eucalyptus [10], or Aneka [11] to manage the local resources. In addition, two interfaces to connect to a Grid middleware (i.e., Grid'5000) and an IaaS Cloud provider (i.e., Amazon's EC2 [12]) have been developed. Moreover, an emulated VIE for testing and debugging has been implemented for VMM.

The persistence database is used for storing information of the gateway such as VM templates and peering arrangements. In this work, we consider the case where the public Cloud provider has a matching VM template for each available template at the database. The Management and Monitoring enables gateway to manage and monitor resources such as Java applications.

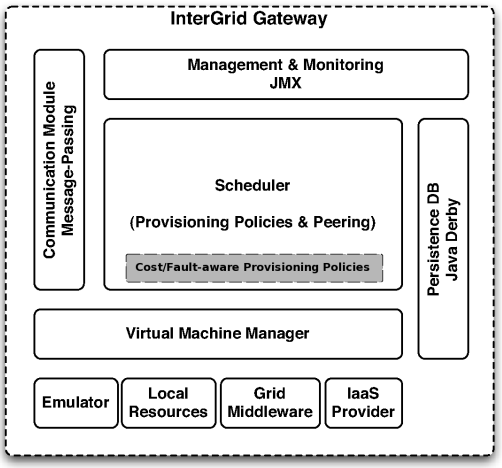


Fig. 2 IGG components.

The Communication Module provides as asynchronous message-passing mechanism, and received messages are handled in parallel by a thread-pool. That makes gateway loosely coupled and allows for more failure-tolerant communication protocols.

2.3 System Workload

In this paper, we consider scientific applications characterized by tightly-coupled computations and communications. Computational Fluid Dynamic (CFD) applications are examples of such applications. Each application can include several tasks and they are sensitive to communication networks in terms of delay and bandwidth. Therefore, they may not benefit heavily from resource co-allocation from multiple provides in virtualized environments [13], and must be served with resources from a single resource provider. In our problem, we define a user's request as corresponding to an individual job while an application may include several jobs. Also, we assume requests are not deadline constrained.

3 Related Work

This section describes the related work pertaining to the utilization of Cloud computing resources and augmenting them with local infrastructure to increase resource availability to solve scientific computing applications.

Several load sharing mechanisms have been proposed for different types of distributed systems. Iosup *et al.* [14] proposed a matchmaking mechanism for enabling resource sharing across computational Grids. Balazinska *et al.*

[15] investigated a mechanism for migrating stream processing operators in a federated distributed system. We address the adaptive resource provisioning which borrow resources from a Cloud provider to improve the reliability and performance of an organization's infrastructure in the presence of resource failures.

In [16] a system of brokers that enable the leasing of various types of resources including virtual machines is provided. Also, authors in [17] investigated the number of migrations required when the broker and a site scheduler use conflicting policies. VioCluster [18] is a system in which a broker is responsible for dynamically managing a virtual domain by borrowing and lending machines between clusters. Our proposed policies in this paper, do not rely on any information from the local scheduler. We also consider the resource failures model to propose adaptive brokering strategies.

Montero *et al.* [19] also used GridWay to deploy virtual machines on a Globus Grid; They also proposed the GridGateWay [20] to enable grid interoperability of Globus Grids. In [1], authors developed virtual infrastructure management through two open source projects, OpenNebula and Haizea[2]. In contrast, we developed the InterGrid environment that is based on the virtual machine technology and can be connected to any distributed systems through Virtual Machine Manager (VMM). Moreover, we consider a new type of platform which is commonly called Hybrid Cloud and propose adaptive brokering strategies which are part of InterGrid Gateway (IGG) to utilize public Cloud resources.

The applicability of public Cloud services for Grid computing has been demonstrated in existing work. In [4], authors consider the Amazon data storage service S3 for scientific data-intensive applications. They conclude that monetary costs are high as the storage service groups availability, durability, and access performance together. In contrast, data-intensive applications often do not need all of these three features. Garfinkel conducts a general cost-benefit analysis of Clouds in [21]. However, no specific type of scientific application is considered. In [3], the authors determine the cost of running a scientific workflow over a Cloud. They find that the computational costs outweighed storage costs for their Montage application. Kondo *et al.* [2] provided cost-benefit analysis of Cloud computing versus desktop grids for compute-intensive tasks. In our work, in contrast to others, we consider the workload as the users' requests which can be data or compute intensive jobs. Also, instead of cost-benefit analysis, we proposed cost-aware brokering strategies to extend the capacity of an unreliable local cluster.

In [22], authors developed a model of an *Elastic Site* that utilized service provided by a site to take advantage of elastically provisioned resources in a public Cloud. The authors in [5] investigated whether an organization using a local cluster can benefit from using Cloud providers to improve the performance of its users' requests. The authors in [23] utilized gang scheduling to dispatch parallel jobs to a cluster of VMs which are hosted at Amazon's

[2] http://haizea.cs.uchicago.edu

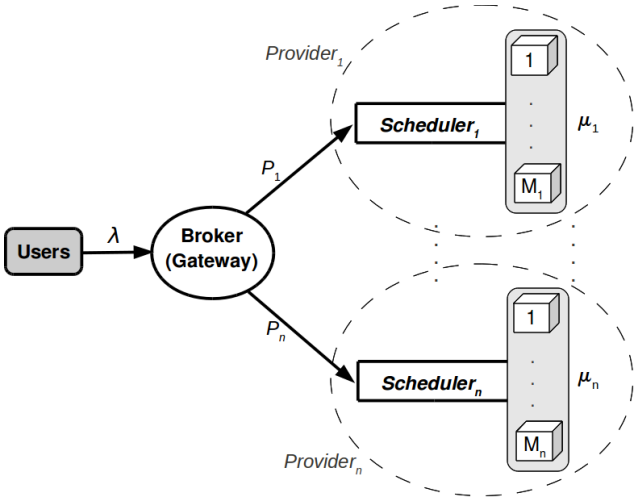


Fig. 3 Model of resource brokering for n providers.

EC2. Our work is different in several aspects. We proposed a general model of resource provisioning for n providers which can be applied to other types of systems and applications. We also proposed cost/fault-aware brokering strategies based on non-observable queues that do not need to have any information about the local queue of providers. In contrast to the work by Assunção *et al.* [5], the proposed policies are independent of the scheduling algorithms. Moreover, we evaluate the performance of the system under realistic workload and failure traces.

4 The Proposed Generic Resource Provisioning

As mentioned earlier, the resource provisioning policy in the system under study has three steps: resource brokering, dispatching, and scheduling of requests. The first two steps are performed by the Gateway (e.g., IGG) and the last step is done at the local scheduler of each resource provider. In this section, we assume n providers and propose a generic solution for the three resource provisioning steps.

4.1 Adaptive Brokering Strategy

We formulate the problem of resource brokering similar to that of routing in distributed parallel queues [24,7]. That is, we consider each resource provider as one server with a given service rate; a scheduler that serves as an input queue; and a broker that redirects the incoming requests to one of the providers as shown in Figure 3.

In the following subsection, the proposed brokering strategy finds the routing probabilities. We assume that the broker located in front of n heterogeneous multi-server parallel queues routes the given requests to providers according to the routing probabilities P_i.

Requests arrive to the broker with a given distribution I, mean $E[I] = \lambda^{-1}$ and variance $V[I] = \sigma_I^2$. We assume the broker holds no queues (we will elaborate on this assumption in Section 4.2). Therefore, requests are handled immediately by the broker upon arrival. Each resource provider is modeled as a single queue with M_i nodes and a local scheduler. Furthermore, we assume service time of queue i follows a given distribution S_i with mean $E[S_i] = \mu_i^{-1}$ and coefficient of variance $C_{S_i} = \sigma_{S_i} \cdot \mu_i$.

Another aspect of the problem that must be taken into account is the *computing cost*. While commercial Cloud is made available in pay-as-you-go manner, computing cost of local infrastructure usually is not easy to estimate, as it depends on many issues such as life time, system utilization, system technology, etc. However, ignoring the cost of a local infrastructure and assuming the resources are free with respect to the Cloud resources is unrealistic. There are some obvious source of costs that can be considered, such as high start-up costs for purchasing hardware or power and cooling costs. Therefore, in this model, we associate a price, K_i, to be paid to each of provider i based on resource usage per time unit. This parameter can be considered as the holding cost, or weight per request per time unit at queue i. K_i can be defined as a constant value or a function of system parameters. For example, in Amazon's EC2, on-demand instances have fixed price while Spot instances have variable price which is a function of VM demand [12].

Considering the associated cost as well as response time of the given requests for each resource provider, the objective function for the broker could be expressed as follows:

$$min \sum_{i=1}^{n} (K_i \cdot E[T_i]) \tag{1}$$

where $E[T_i]$ is the expected response time of requests served at queue i and is described in the following.

Based on Figure 3, queue i has the mean inter-arrival time $E[I_i] = \lambda_i^{-1} = (P_i\lambda)^{-1}$, so we can find its variance by Wald's equation [25] as follows:

$$V[I_i] = \frac{\sigma_I^2 P_i + \lambda^{-2}(1 - P_i)}{P_i^2} \tag{2}$$

As mentioned before, queue i has the mean service time and the coefficient of variance μ_i^{-1} and $\sigma_{S_i} \cdot \mu_i$, respectively. As the incoming requests have several VMs that potentially can be as large as M_i nodes, we model each provider with a single server queue. By considering general distribution for the inter-arrival time as well as the service time, each queue can be modeled as a GI/GI/1 FCFS queue. Therefore, we are able to approximate the expected response

time of queue i by the following equation[3] [24]:

$$E[T_i] = \frac{1}{\mu_i} + \frac{C_{I_i}^2 - C_{S_i}^2}{2(\mu_i - \lambda_i)} \tag{3}$$

where $C_{I_i}^2$ is the squared coefficient of variance for the inter-arrival time at queue i, and can be calculated using Equation (2) as follows:

$$C_{I_i}^2 = \frac{V[I_i]}{E[I_i]^2} = 1 + P_i(\lambda^2\sigma_I^2 - 1) \tag{4}$$

Next, we apply Lagrange multipliers method to optimize Equation (1) using $E[T_i]$ from Equation (3) and assuming $\sum_{i=1}^{n} P_i = 1$. The solution of this optimization gives the routing probabilities P_i as follows:

$$P_i = \frac{\mu_i}{\lambda} - \frac{\sum_{i=1}^{n} \mu_i - \lambda}{\lambda} \cdot \frac{\sqrt{K_i\eta_i}}{\sum_{i=1}^{n} \sqrt{K_i\eta_i}} \tag{5}$$

where η_i can be calculated by the following equation:

$$\eta_i = \lambda(\lambda^2\sigma_I^2 + \lambda^2 + C_{S_i}^2 - \lambda\mu_i) \tag{6}$$

Equation (5) reflects the effect of the system parameters as well as computing costs on the routing probabilities leading to a *cost-aware* brokering strategy. Moreover, the proposed brokering strategy is based on non-observable queues which means it does not need any information about the scheduler's queues. This simplifies implementation of the IGG in the hybrid Cloud system.

4.2 Dispatch Sequences

The proposed adaptive brokering strategy in the previous subsection determines only the routing probabilities (i.e., P_i). However, it does not explain any sequence for dispatching the incoming requests to the resource providers. Here, we consider two dispatch sequences including probabilistic and deterministic methods to complete the second step of resource provisioning.

Given the routing probabilities, one way to dispatch the requests is using a *Bernoulli* distribution to randomly submit the requests. In this case, the gateway only uses routing probabilities without any special sequencing of requests sent to providers. In this sense, this method is memoryless as it does not take into account which requests have been sent to which queues. We call this method Adaptive with Random Sequence (ARS) policy.

In contrast, we also propose another method with deterministic sequence, which considers the past sequence of dispatching with a very limited time overhead. This method, we call as Adaptive with Deterministic Sequence

[3] There are several approximations for this queue in the literature, but we choose one which is a good estimate for heavily loaded systems.

(ADS) policy. To generate the deterministic sequence, we used the *Billiard* scheme [26], determined as follows.

Suppose a billiard ball bounces in an n-dimensional cube where each side and opposite side are assigned by an integer value in the range of $\{1, 2, ..., n\}$. Then, a deterministic billiard sequence is generated by a series of integer values which shows the sides hit by the ball when shot. In [26], the authors proposed a method to generate the billiard sequence as follows:

$$i_b = \min_{\forall i} \left\{ \frac{X_i + Y_i}{P_i} \right\} \tag{7}$$

where i_b is the *target* queue, and X and Y are vectors of integers with size n. X_i reflects the fastest queue, and is set to one for the fastest queue and zero for all other queues [7]. Y_i keeps track of the number of requests that have been sent to queue i and is initialized to zero. After finding the target queue, it is updated as $Y_{i_b} = Y_{i_b} + 1$. P_i is the fraction of requests that are sent to queue i and is the same as the routing probabilities obtained from Equation (5).

Based on the proposed methods for dispatching, the assumption about the broker without a queue would be justifiable as the broker has only a few computation operations to make decision about target providers for incoming requests.

4.3 Scheduling Algorithms

The last step in the resource provisioning is scheduling of request on the available VMs in the resource providers. For this purpose, we utilize three well-known scheduling algorithms *conservative* [27], *aggressive* [28], and *selective* backfilling [29]. With conservative backfilling, each request is scheduled when it is submitted to the system, and requests are allowed to leap forward in the queue if they do not delay other queued requests. In aggressive backfilling (EASY), only the request at the head of the queue, called *the pivot*, is granted a reservation. Other requests are allowed to move ahead in the queue as long as they do not delay the pivot. Selective backfilling grants reservation to a request when its expected slowdown exceeds a threshold. This implies, the request has waited long enough in the queue. The expected slowdown of a given request is also called *eXpansion Factor* (XFactor) and is given by the following equation:

$$XFactor = \frac{w_j + d_j}{d_j} \tag{8}$$

where w_j and d_j are the waiting time and the run time of request j, respectively. We use the Selective-Differential-Adaptive scheme proposed in [29], which lets the XFactor threshold be the average slowdown of previously completed requests.

We assume that each VM runs on one available node. As a given request needs *all* VMs to be available for the whole required duration, any failure

event in any virtual machine would stop execution of the whole request. The request can be started again, if and only if all VMs become available again. If there is a resource failure during execution we apply *checkpointing* [30] technique to resume execution of the request from where it was interrupted. We incorporate checkpointing in our scheduling algorithms and provide a fault-tolerant environment for serving requests in the local cluster.

5 Case Study: Hybrid Cloud With Two Providers

In this section, we adopt the results of Section 4 for our specific case where we have two providers (i.e., $n = 2$). We use index $i = s$ for the local cluster and $i = c$ for the Cloud hereafter. Moreover, we assume that there is computing speed homogeneity within each provider. As mentioned earlier, the proposed policies are part of the IGG (see Figure 2).

To apply the proposed analytical model for brokering strategy, we first need to specify the arrival distribution I. The arrival distribution I depends on the system workload and could be given as a general distribution with light-tails [24]. As can be seen from Equation (3), the mean service time, μ_i, and coefficient of variance, C_{S_i} are two unknown parameters. Therefore, in the following, we determine μ_s and C_{S_s} for the local cluster and μ_c and C_{S_c} for Cloud to obtain the corresponding routing probabilities by Equation (5).

5.1 Runtime Model for Local Cluster

The distribution of service time in each provider depends on the characteristics of the infrastructure as well as the input workload. Moreover, in our analysis in Section 4, *relative* response times are more important than absolute response times. The reason is that scaling up or down of the service times in Equation (1) does not change the routing probabilities.

Since we assume an unreliable local cluster with resource failures, we must consider the availability and unavailability intervals of each resource to work out the service time distribution. We term the continuous period of a service outage due to a failure as an *unavailability interval*. A continuous period of availability is called an *availability interval*. For this purpose, we use the proposed model by Kleinrock *et al.* [31] to find the mean and coefficient of variance of completion time for W time units of work over M transient processors, as follows:

$$\overline{f} = \frac{W}{\overline{b}} = \frac{W}{M}\frac{(t_a + t_u)}{t_a} \tag{9}$$

$$\frac{\sigma_f}{\overline{f}} = \frac{\sqrt{\sigma_b^2}}{\sqrt{\overline{b}W}} \tag{10}$$

where

$$\overline{b} = \frac{t_a}{(t_a + t_u)}M \tag{11}$$

$$\sigma_b^2 = \frac{\sigma_a^2 t_u^2 + \sigma_u^2 t_a^2}{(t_a + t_u)^3} M \tag{12}$$

Moreover, t_a, t_u, σ_a^2 and σ_u^2 are the mean and the variance of availability and unavailability interval lengths, respectively.

As the input workload includes parallel requests (i.e., request with several VMs), we consider the mean request size ($\overline{W}$) as the given work to the system. We define the mean request size by multiplying the mean number of VMs ($\overline{V}$) by the mean request duration ($\overline{D}$). Hence, $\overline{W} = \overline{V} \cdot \overline{D}$. These two parameters are dependent on workload model (see Section 6.1).

By considering $\overline{W}$ time units of work over M_s failure-prone nodes, we define the service rate of the cluster queue as the reciprocal value of the mean completion time for a given workload as follows:

$$\mu_s = \left(\frac{\overline{W}}{M_s \cdot \tau_s} \frac{t_a + t_u}{t_a} + L_s \right)^{-1} \tag{13}$$

where τ_s is the computing speed of the nodes in the local cluster in terms of instruction per second, and L_s is the time to transfer the application (e.g., configuration file or input file(s)) to the cluster through the communication network. Another required parameter is the coefficient of variance of the cluster' service time (i.e., C_{S_s}) which is nothing but Equation (10). This makes our brokering strategy *failure-aware* and consequently adaptive to the system's failure pattern.

5.2 Runtime Model for Public Cloud

In this work, we investigate a hybrid Cloud system including an unreliable local cluster while public Cloud is able to provide *highly reliable* services to their customers [32]. Although resource failures are inevitable, but public Cloud providers adopt carefully engineered modules include *redundant* components to cope with resource failures [33]. This design style is too expensive to consider for design and implementation of local clusters.

Therefore, we can use Normal distribution for the request completion time in the Cloud. This can be justified by the central limit theorem which assures that when summing many independent random variables (here requests completion time), the resulting distribution tends toward a Normal distribution. So, the service rate of the Cloud queue can be found as the reciprocal values of the mean request completion time for a given workload on M_c reliable nodes as follows:

$$\mu_c = \left(\frac{\overline{W}}{M_c \cdot \tau_c} + L_c \right)^{-1} \tag{14}$$

where τ_c and L_c are the computing speed and the time to transfer the application to public Cloud provider, respectively. It should be noted that the time

to transfer output data to the local cluster is not considered as it can be overlapped with other computations. The coefficient of variance of the service time can be assumed as one (i.e., $C_{S_c} = 1$) to model the *performance variability* in public Cloud resources [34]. This can be the minimum value for the coefficient of variance and should be increased on the basis of variance in performance of Cloud resources. Moreover, this is the parameter that should be changed to adapt the proposed performance model for different types of resources in a public Cloud provider (e.g., different instances in Amazon's EC2 [12]).

Apart from the brokering strategy, other two steps of resource provisioning can be directly used from Section 4. For $n = 2$, $X_c = 1$ and $X_s = 0$ in the billiard scheme, for our specific case.

6 Performance Evaluation

In order to evaluate the performance of the proposed policies, we implemented a discrete event simulator using CloudSim [35]. We used simulation as experiments are reproducible and the cost of conducting experiments on a real public Cloud would be prohibitively expensive.

The performance metrics related to response times of requests that are considered in all simulation scenarios are the Average Weighted Response Time (AWRT) [36] and the bounded slowdown [37]. The AWRT for N given requests is defined by the following equation:

$$AWRT = \frac{\sum_{j=1}^{N} d_j \cdot v_j \cdot (ct_j - st_j)}{\sum_{j=1}^{N} d_j \cdot v_j} \tag{15}$$

where v_j is the number of virtual machines of request j. ct_j is the time of completion of the request and st_j is its submission time. The resource consumption ($d_j \cdot v_j$) of each request j is used as the weight. The AWRT measures the average time that users must wait to have their requests completed. The bounded slowdown metric, is defined as follows:

$$Slowdown = \frac{1}{N} \sum_{j=1}^{N} \frac{w_j + max(d_j, bound)}{max(d_j, bound)} \tag{16}$$

where w_j is the waiting time of request j. Also, *bound* is set to 10 seconds to eliminate the effect of very short requests [37].

We evaluate the proposed policies against another basic policy, the No-Redirection policy. This is the simplest brokering policy with the routing probability of the local cluster set to one ($P_s = 1$) and set to zero for Cloud ($P_c = 0$). In this policy, all requests run only on the unreliable local cluster.

Table 1 Input parameters for the workload model.

Parameters	Distribution/Value
Inter-arrival time	Weibull ($\alpha = 23.375, 0.2 \leq \beta \leq 0.3$)
No. of VMs	Loguniform ($l = 0.8, m = 3.5, h = 6, q = 0.9$)
Request duration	Lognormal ($2.5 \leq \theta \leq 3.5, \sigma = 1.7$)
P_{one}	0.02
P_{pow2}	0.78

6.1 Workload Model

The workload model for evaluation scenarios is obtained from the Grid Workload Archive [38]. We used the parallel job model of the DAS-2 system which is a multi-cluster Grid [39]. Based on the workload characterization, the inter-arrival time, request size, and request duration follow Weibull, two-stage Loguniform and Lognormal distributions, respectively. These distributions with their parameters are listed in Table 1. It should be noted that the number of VMs in the request can be scaled to the system size (e.g., M nodes) by setting $h = log_2 M$.

To find the mean number of VMs per request, we need the probability of different number of VMs in the incoming requests. Assume that P_{one} and P_{pow2} are probabilities of request with one VM and power of two VMs in the workload, respectively. Therefore, the mean number of VMs required by requests is given as follows:

$$\overline{V} = P_{one} + 2^{\lceil r \rceil}(P_{pow2}) + 2^r\left(1 - (P_{one} + P_{pow2})\right) \tag{17}$$

where r is the mean value of the two-stage uniform distribution with parameters (l, m, h, q) as listed in Table 1 and can be found as follows:

$$r = \frac{ql + m + (1-q)h}{2} \tag{18}$$

Additionally, the mean request duration is the mean value of the Lognormal distribution with parameters (θ, σ) which is given by:

$$\overline{D} = e^{\theta + \frac{\sigma^2}{2}} \tag{19}$$

6.2 Experimental Setup

For each simulation experiment, statistics were gathered for a two-month period of the DAS-2 workloads. The first week of workloads during the *warm-up* phase were ignored to avoid bias before the system reached steady-state. For the experiments, each data point is the average of 30 simulation rounds including several jobs vary from 3,000 to 25,000 (depends on the workload parameters). In our experiments, the results of simulations are accurate within a confidence level of 95%.

Table 2 Input parameters for the failure model.

Parameters	Description	Value (hours)
t_a	Mean availability length	22.25
σ_a	Std of availability length	41.09
t_u	Mean unavailability length	10.22
σ_u	Std of unavailability length	40.75

The number of resources in the local cluster and Cloud is equal to $M_s = M_c = 64$ with homogeneous computing speed[4] (i.e., $\tau_s = \tau_c = 1000$ MIPS). Moreover, the cost of resources in the Cloud is considered to be five times more expensive than the local cluster's resources (i.e., $K_s = 1$, $K_c = 5$). The network transfer time of the cluster is negligible as the local resources are interconnected by a high-speed network, $L_s = 0$. However, to execute the application on the public Cloud we must send the configuration file as well as input file(s). Therefore, we consider the network transfer time as $L_c = 64$ sec., which is the time to transfer 80 MB data[5] on a 10 Mbps network connection [6].

The failure trace for the experiments is obtained from the Failure Trace Archive [6]. We used the failure trace of a cluster in the Grid'5000 with 64 nodes for duration of 18 months, which includes on average 795 events per node. An event is defined as a failure event (when the status of a resource changes from availability to unavailability) or a recovery event (when the status of a resource changes from unavailability to availability). The parameters for the failure model of Grid'5000 are listed in Table 2 (see [6] for more details). Also, each experiment utilizes a unique starting point in the failure traces to avoid bias results.

In order to generate different synthetic workloads, we modified two parameters of the workload model, one at a time. To change the inter-arrival time, we modified the second parameter of the Weibull distribution (the *shape* parameter β) as shown in Table 1. Also, to have requests with different duration, we changed the first parameter of the Lognormal distribution between 2.5 and 3.5 which is mentioned in Table 1.

To compute the cost of using resources from a public Cloud provider, we use the amounts charged by Amazon to run basic virtual machines and network usage at EC2. For the total of N requests which are submitted to the system, the cost of using EC2 can be calculated as follows:

$$Cost_{EC2} = (H_c + N \cdot P_c \cdot H_u)\, C_p + (N \cdot P_c \cdot B_{in})\, C_x \tag{20}$$

where H_c is the total Cloud usage per hour. This implies, if a request uses a VM for 40 minutes for example, the cost of one hour is considered. $N \cdot P_c$ is

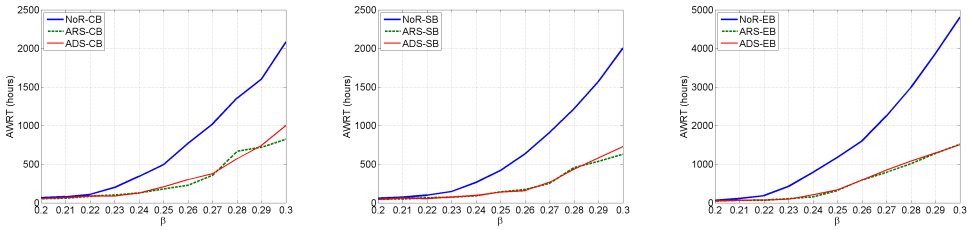
[4] This assumption is made just to focus on performance degradation due to failure.

[5] This is the maximum amount of data for a real scientific workflow application [40].

[6] The network latency is negligible as it is less than a second for public Cloud environments [41].

the fraction of requests which are redirected to the public Cloud. Also, H_u is the startup time for initialization of operating system on a virtual machine which is set to 80 seconds [34]. We take into account this value as Amazon commences charging users when the VM process starts. B_{in} is the amount of data which transfer to Amazon's EC2 for each request and as it is mentioned before, it is 80 MB per request. The cost of one specific instance on EC2 (us-east) is determined as C_p and considered as 0.085 USD per virtual machine per hour for a small instance. The cost of data transfer to Amazon's EC2 is also considered as C_x which is 0.1 USD per GB [7]. It should be noted that we consider a case where requests' output are very small and can be transfered to the local cluster for free [12].

6.3 Results and discussions



(a) Conservative Backfilling (b) Selective Backfilling (c) EASY Backfilling

Fig. 4 AWRT versus arrival rate ($\theta = 3.0$).

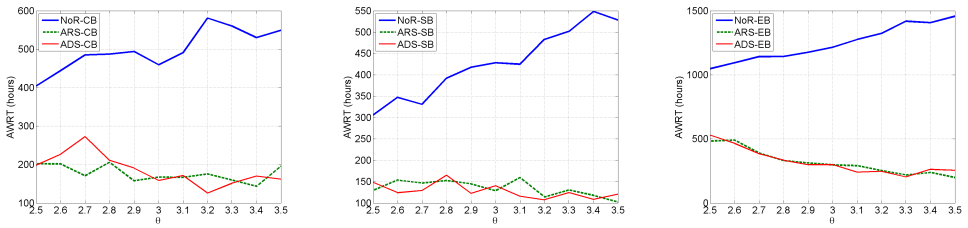
In this section, NoR, ARS, and ADS refer to the No-Redirection, Adaptive with Random Sequence, and Adaptive with Deterministic Sequence, respectively. Moreover, CB, SB, and EB stand for Conservative, Selective and EASY Backfilling, respectively. The same scheduling algorithms are used for the local cluster and Cloud in all scenarios.

The simulation results for AWRT versus arrival rate are depicted in Figure 4 for different provisioning policies while average request duration is kept of medium size (i.e., $\theta = 3.0$). For all these cases, we see that increasing the arrival rate dramatically increases the request failure rate for AWRT in NoR policy. On the other hand, the ARS and ADS policies control the failure rate by redirecting the requests to the Cloud which lead to lower AWRT. The maximum improvement factor of using adaptive brokering with respect to NoR is 3.4, 3.7, and 5.1 times in terms of AWRT for conservative, selective and EASY backfilling, respectively. Although, ADS uses deterministic sequence, there is probability of changing this sequence due to request backfilling in the local

[7] All prices obtained at time of writing this paper during May-June 2011.

scheduler. Therefore, the ADS achieves almost the same performance as the ARS for all the scheduling algorithms.

Figure 5 shows AWRT against different request duration in the moderate arrival rate (i.e., $\beta = 0.25$). It reveals that ADS policy is slightly better than ARS for selective scheduling algorithm. To be more precise, the average performance improvement of ADS with respect to the ARS is 5.8% for selective backfilling. In the case of conservative and EASY backfilling there is no considerable improvement. The reason to have some fluctuations in these figures is the effect of backfilling in the scheduler queue due to changing of requests duration.



(a) Conservative Backfilling (b) Selective Backfilling (c) EASY Backfilling

Fig. 5 AWRT versus request duration ($\beta = 0.25$).

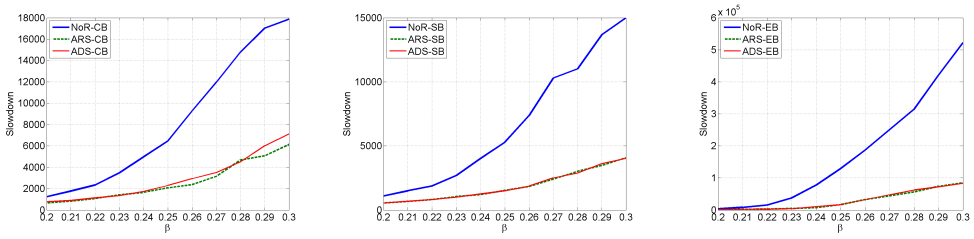
Figure 6 expresses slowdown of requests versus arrival rate for different provisioning policies with the same configuration as previous experiments. Based on these figures, use of adaptive brokering strategies decreases the request slowdown by 4 times for conservative and selective backfilling and 10.9 times for EASY backfilling with respect to NoR policy. As it is illustrated, by increasing the arrival rate, the gap between NoR and adaptive brokering strategies increases. This is because the routing probabilities are strongly dependent on the arrival rate and adaptively redirect more requests to the Cloud to control the system performance.

In Figure 7, request slowdown is depicted against various request sizes for all three policies. These figures reveal that by increasing the request duration, the slowdown decreases while the ADS marginally surpasses ARS for the selective and EASY backfilling. For the conservative backfilling the performance of ARS is almost better than ADS, specially for short request duration.

It is worth noting that the theoretical work in [7] showed that the Billiard scheme provides optimal response time when all queues are FCFS and service times follow the exponential distribution. However, in our case, queues are not FCFS due to probability of backfilling and service times are not simple short-tailed distributions. We observed that in this situation, the ADS policy with billiard sequence is not able to perform very well with respect to the ARS due to perturbation of sequence in the scheduler of the resource providers.

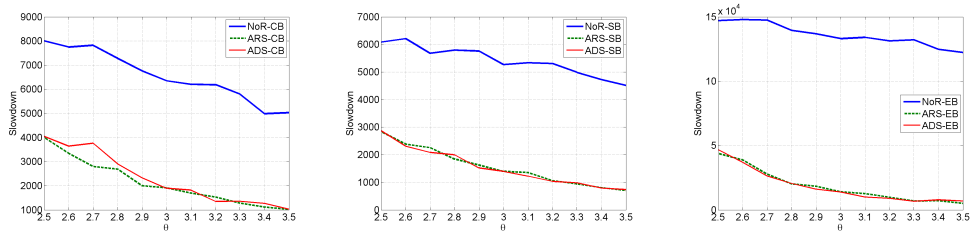
It should be noted that, we do not consider the checkpointing overheads in this study, as we want to focus on effect of failures in the resource provision-

ing. In other words, we show that even without time and space overheads of checkpointing mechanism, resource provisioning from a public Cloud substantially improve the system performance. This improvement would be the lower bound of enhancement while including overheads of checkpointing.



(a) Conservative Backfilling (b) Selective Backfilling (c) EASY Backfilling

Fig. 6 Slowdown versus arrival rate ($\theta = 3.0$).



(a) Conservative Backfilling (b) Selective Backfilling (c) EASY Backfilling

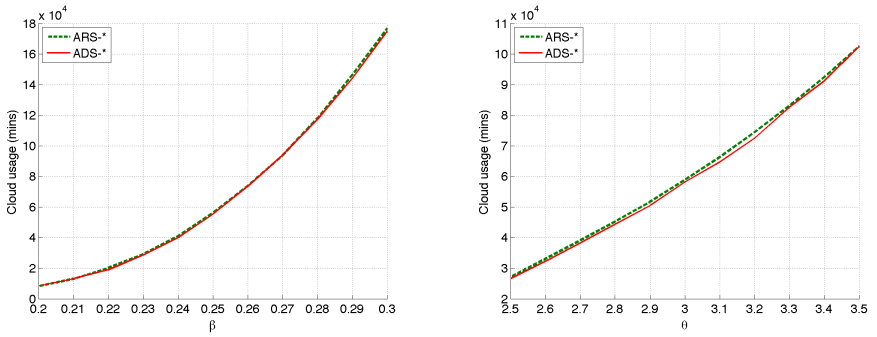
Fig. 7 Slowdown versus request duration ($\beta = 0.25$).

6.4 Cost analysis on a public Cloud

To analyze how the proposed policies utilize the Cloud resources, Figure 9 shows the amount of Cloud usage in time versus arrival rate (Figure 8(a)) and request duration (Figure 8(b)) for the two-month long workload.

We observed that the amount of Cloud usage (H_c in Equation (20)) for different scheduling algorithms is exactly the same. However, as we illustrated before in this section, the proposed policies have different performance in terms of AWRT and slowdown. Moreover, this confirms that our proposed policies are independent from the scheduling algorithms. Recall that in the objective function of the broker in Equation (1), both cost and response time can be *relative*, so the monetary cost of Cloud usage is independent from the broker' cost parameters (K_i).

Nevertheless, according to the experiments the ADS utilizes up to 6.7% less resources from the Cloud provider with respect to the ARS. This can save some amount of money that must be paid for utilizing of the Cloud resources. To be more precise, in the following, we discuss how we can map this usage to a real public Cloud provider (i.e., Amazon's EC2).

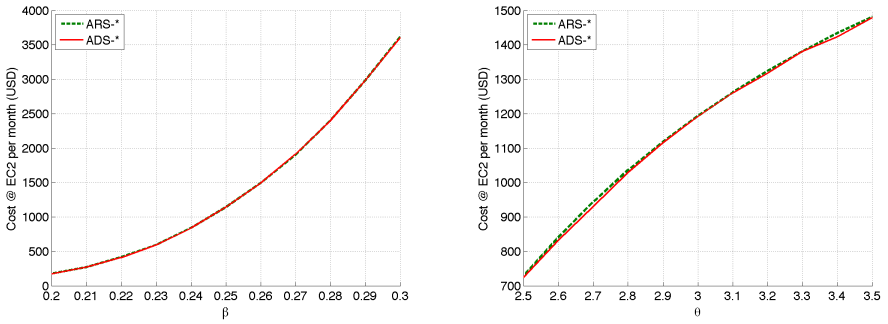


(a) Various arrival rate ($\theta = 3.0$) (b) Various request duration ($\beta = 0.25$)

Fig. 8 Amount of Cloud usage for all provisioning polices (* stands for CB, SB, and EB).

To do this, we calculated the usage of EC2 and its associated cost per month using Equation (20). Figure 9(a) and Figure 9(b) show the amount of money spent on EC2 per month to respond to the incoming requests with different arrival rate and request duration, respectively. As noted before, the Cloud usage for all proposed policies are the same because of the cost-time optimization in the broker. Moreover, ADS and ARS have the same monetary cost on EC2 for various arrival rate while ADS incurs lower cost for various duration size (see Figure 9(b)). To show the difference of the proposed policies in terms of cost and performance under different working conditions, we illustrate a quantitative analysis.

Table 3 lists the performance improvement all proposed polices with respect to the case of using only the unreliable local cluster. In each row the values are presented for different request duration while the input load is moderate (i.e., $\beta = 0.25$). In addition, the Cloud cost of each scenario is presented in Table 4. We can observe that with a limited cost (e.g., less than 1200 USD) per month, we can improve the performance of users' requests up to 4.10 in terms of AWRT and up to 9.58 in terms of slowdown. However, by spending more money per month, we are able to obtain up to 5.90 and 17.61 times improvement in terms of AWRT and slowdown, respectively. As it is illustrated in Table 3, the EASY backfilling improves dramatically in terms of AWRT and slowdown for medium and large requests. Additionally, in almost all cases, the ADS policy is slightly better than ARS policy in terms of the cost at EC2 per month.



(a) Various arrival rate ($\theta = 3.0$) (b) Various request duration ($\beta = 0.25$)

Fig. 9 Cost of using EC2 per month for various request duration and a moderate load ($\beta = 0.25$) (* stands for CB, SB, and EB).

Table 3 The performance improvement of provisioning polices for different request durations and moderate input load ($\beta = 0.25$, A: AWRT, S: Slowdown).

Request duration	Conservative				Selective				EASY			
	ARS		ADS		ARS		ADS		ARS		ADS	
	A	*S*	*A*	*S*	*A*	*S*	*A*	*S*	*A*	*S*	*A*	*S*
Short ($\theta = 2.5$)	2.00	1.99	2.04	1.98	2.37	2.14	2.06	2.12	2.16	3.35	1.98	3.13
Medium ($\theta = 3.0$)	2.75	3.13	2.89	3.35	3.33	3.78	3.05	3.78	4.10	9.44	4.06	9.58
Large ($\theta = 3.4$)	3.69	4.46	3.11	3.93	4.65	5.88	5.05	5.97	5.90	17.61	5.36	15.97

Table 4 The Cloud cost of provisioning polices for different request durations and moderate input load ($\beta = 0.25$).

Request duration	EC2 cost/month (USD)	
	ARS	ADS
Short ($\theta = 2.5$)	728.40	724.20
Medium ($\theta = 3.0$)	1193.60	1191.60
Large ($\theta = 3.4$)	1434.40	1423.80

7 Conclusions

We considered the problem of Cloud computing resource provisioning to extend the computing capacity and performance of an unreliable local cluster. We presented a generic resource provisioning model based on the stochastic analysis of routing in distributed parallel queues where the arrival and service processes follow general distributions. The proposed brokering strategy is adaptive to the cost and response time of resource providers. Both proposed policies, ARS and ADS, utilize adaptive brokering strategy while ARD adopts probabilistic sequence and ADS uses deterministic sequence to redirect the requests. The proposed policies take advantage of non-observable queues, so they do not require any information about the scheduler's queues.

Experimental results under realistic workload and failure events reveal that both policies reduces AWRT and slowdown of requests significantly for different scheduling algorithms, where ADS policy shows marginally better cost

than ARS. We observed that request backfilling strongly modifies the sequence of requests in the queues, so the ADS policy can not achieve a considerable improvement with respect to ARS policy. Finally, we believe that the proposed performance model can be a practical evaluation tool that can help system administrators to explore the design space and examine various system parameters.

In future work, we intend to consider deadline-constrained requests and evaluate the effect of Cloud computing resource provisioning on such requests. In addition, moving VMs between local resources and public Cloud will be another approach to deal with resource failures in the local clusters, for the loosely-coupled parallel applications. We also intend to implement the proposed strategies inside the IGG and run real experiments. For this purpose, we will investigate different checkpointing mechanisms in our analysis and implementation as well.

Acknowledgements

The authors would like to thank Jonatha Anselmi, Rodrigo N. Calheiros, Mohsen Amini, and Amir Vahid for useful discussions.

References

1. B. Sotomayor, R. S. Montero, I. M. Llorente, I. Foster, Virtual infrastructure management in private and hybrid clouds, IEEE Internet Computing 13 (5) (2009) 14 –22.
2. D. Kondo, B. Javadi, P. Malecot, F. Cappello, D. P. Anderson, Cost-benefit analysis of Cloud computing versus desktop grids, in: Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009), IEEE Computer Society, Washington, DC, Rome, Italy, 2009, pp. 1–12.
3. E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, The cost of doing science on the Cloud: The montage example, in: Proceedings of the 19th ACM/IEEE International Conference on Supercomputing (SC 2008), IEEE Press, Piscataway, NJ, Austin, Texas, 2008, pp. 1–12.
4. M. R. Palankar, A. Iamnitchi, M. Ripeanu, S. Garfinkel, Amazon S3 for science Grids: a viable solution?, in: Proceedings of the 1st International Workshop on Data-aware Distributed Computing (DADC'08) in conjunction with HPDC 2008, ACM, New York, NY, Boston, MA, 2008, pp. 55–64.
5. M. D. de Assunção, A. di Costanzo, R. Buyya, Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters, in: Proceedings of the 18th International Symposium on High Performance Parallel and Distributed Computing (HPDC 2009), ACM, New York, NY, Garching, Germany, 2009, pp. 141–150.
6. D. Kondo, B. Javadi, A. Iosup, D. H. J. Epema, The Failure Trace Archive: Enabling comparative analysis of failures in diverse distributed systems, in: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010), IEEE Computer Society, Washington, DC, Melbourne, Australia, 2010, pp. 398–407.
7. J. Anselmi, B. Gaujal, Optimal routing in parallel, non-observable queues and the price of anarchy revisited, in: 22nd International Teletraffic Congress (ITC), Amsterdam, The Netherlands, 2010.
8. A. di Costanzo, M. D. de Assunção, R. Buyya, Harnessing cloud technologies for a virtualized distributed computing infrastructure, IEEE Internet Computing 13 (5) (2009) 24–33.

9. J. Fontán, T. Vázquez, L. Gonzalez, R. S. Montero, 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, CA, 2008.
10. D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The Eucalyptus open-source cloud-computing system, in: Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2009), IEEE Computer Society, Washington, DC, Shanghai, China, 2009, pp. 124–131.
11. C. Vecchiola, X. Chu, R. Buyya, Aneka: A Software Platform for .NET-based Cloud Computing, IOS Press, Amsterdam, 2009, pp. 267–295.
12. Amazon Inc., Amazon Elastic Compute Cloud (Amazon EC2), `http://aws.amazon.com/ec2`.
13. M. Tatezono, N. Maruyama, S. Matsuoka, Making wide-area, multi-site MPI feasible using Xen VM, in: Proceedings of the 4th Workshop on Frontiers of High Performance Computing and Networking in conjuction with ISPA 2006, Springer-Verlag, Berlin, Sorrento, Italy, 2006, pp. 387–396.
14. A. Iosup, D. H. J. Epema, T. Tannenbaum, M. Farrellee, M. Livny, Inter-operating Grids through delegated matchmaking, in: Proceedings of the 18th ACM/IEEE Conference on Supercomputing (SC 2007), ACM, New York, NY, Reno, Nevada, 2007, pp. 1–12.
15. M. Balazinska, H. Balakrishnan, M. Stonebraker, Contract-based load management in federated distributed systems, in: Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004), USENIX Association, Berkeley, CA, San Francisco, CA, 2004, pp. 197–210.
16. D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, K. G. Yocum, Sharing networked resources with brokered leases, in: Proceedings of the USENIX Annual Technical Conference, USENIX Association, Berkeley, CA, Boston, MA, 2006, pp. 199–212.
17. L. Grit, D. Inwin, A. Yumerefendi, J. Chase, Virtual machine hosting for networked clusters: Building the foundations for 'autonomic' orchestration, in: Proceedings of the 1st International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006), IEEE Computer Society, Washington, DC, Tampa, Florida, 2006, pp. 7–15.
18. P. Ruth, P. McGachey, D. Xu, VioCluster: Virtualization for dynamic computational domain, in: Proceedings of the 7th IEEE International Conference on Cluster Computing (Cluster 2005), IEEE Press, Piscataway, NJ, Burlington, MA, 2005, pp. 1–10.
19. A. J. Rubio-Montero, E. Huedo, R. S. Montero, I. M. Llorente, Management of virtual machines on Globus Grids using GridWay, in: Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007), IEEE Press, Piscataway, NJ, Long Beach, USA, 2007, pp. 1–7.
20. E. Huedo, R. S. Montero, I. M. Llorente, Grid architecture from a metascheduling perspective, IEEE Computer 43 (7) (2010) 51 –56.
21. S. Garfinkel, Commodity grid computing with Amazons S3 and EC2, USENIX LOGIN 32 (1) (2007) 7–13.
22. P. Marshall, K. Keahey, T. Freeman, Elastic site: Using clouds to elastically extend site resources, in: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010), IEEE Computer Society, Washington, DC, Melbourne, Australia, 2010, pp. 43–52.
23. I. Moschakis, H. Karatza, Evaluation of gang scheduling performance and cost in a cloud computing system, The Journal of Supercomputing 1 (2010) 1–18.
24. X. Guo, Y. Lu, M. S. Squillante, Optimal probabilistic routing in distributed parallel queues, SIGMETRICS Perform. Eval. Rev. 32 (2) (2004) 53–54.
25. S. M. Ross, Stochastic Processes, second edition, John Wiley and Sons, 1997.
26. A. Hordijk, D. van der Laan, Periodic routing to parallel queues and billiard sequences, Mathematical Methods of Operations Research 59 (2004) 173–192.
27. A. W. Mu'alem, 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) (2001) 529–543.
28. D. A. Lifka, The ANL/IBM SP scheduling system, in: Proceedings of the 1st Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '95), Springer-Verlag, London, Santa Barbara, CA, 1995, pp. 295–303.

29. S. Srinivasan, R. Kettimuthu, V. Subramani, P. Sadayappan, Selective reservation strategies for backfill job scheduling, in: Proceedings of the 8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '02), Springer-Verlag, London, Edinburgh, Scotland, UK, 2002, pp. 55–71.
30. M. Bouguerra, T. Gautier, D. Trystram, J.-M. Vincent, A flexible checkpoint/restart model in distributed systems, in: Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics (PPAM 2010), Springer-Verlag, Berlin, Torun, Poland, 2010, pp. 206–215.
31. L. Kleinrock, W. Korfhage, Collecting unused processing capacity: An analysis of transient distributed systems, IEEE Transactions on Parallel and Distributed Systems 4 (5) (1993) 535–546.
32. J. Varia, Best Practices in Architecting Cloud Applications in the AWS Cloud, Wiley Press, Hoboken, NJ, 2011, pp. 459–490.
33. U. Hoelzle, L. A. Barroso, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Morgan and Claypool Publishers, San Rafael, CA, 2009.
34. S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, A performance analysis of EC2 Cloud computing services for scientific computing, in: Proceedings of the 1st International Conference on Cloud Computing (CloudComp 2009), Springer-Verlag, Berlin, Beijing, China, 2009, pp. 115–131.
35. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of Cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.
36. C. Grimme, J. Lepping, A. Papaspyrou, Prospects of collaboration between compute providers by means of job interchange, in: 13th Job Scheduling Strategies for Parallel Processing, Vol. 4942 of Lecture Notes in Computer Science, Berlin / Heidelberg, 2008, pp. 132–151.
37. D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, P. Wong, Theory and practice in parallel job scheduling, in: Proceedings of the 3rd International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '97), Springer-Verlag, London, Seattle, WA, 1997, pp. 1–34.
38. A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D. H. J. Epema, The Grid Workloads Archive, Future Generation Computer Systems 24 (7) (2008) 672–686.
39. H. Li, D. Groep, L. Wolters, Workload characteristics of a multi-cluster supercomputer, in: Proceedings of the 10th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '04), Springer-Verlag, Berlin, New York, USA, 2004, pp. 176–193.
40. S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. E. Dobson, K. Chiu, A grid workflow environment for brain imaging analysis on distributed systems, Concurrency and Computation: Practice and Experience 21 (16) (2009) 2118–2139.
41. CloudHarmony, `http://cloudharmony.com/`.