2012 IEEE 14th International Conference on High Performance Computing and Communications

Cloud Resource Provisioning to Extend the Capacity of Local Resources in the Presence of Failures

Bahman Javadi
School of Computing, Eng. and Mathematics
University of Western Sydney, Australia
Email: b.javadi@uws.edu.au

Parimala Thulasiraman
IDEAS Laboratory
Dept. of Computer Science
University of Manitoba, Canada
Email: thulasir@cs.umanitoba.ca

Rajkumar Buyya
CLOUDS Laboratory
Dept. of Computing and Info. Systems
The University of Melbourne, Australia
Email: raj@csse.unimelb.edu.au

***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 a 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.**

***Keywords*-Resource provisioning; Cloud computing; Failure-prone clusters; Request scheduling; Brokering.**

I. 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 availability 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* [24], the system used in this paper. High performance computing applications can leverage from such systems to execute data intensive applications for increased performance gain.

In the literature, several works [5], [6], [16], [21] 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 a 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. 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. 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. We define a failure as an event in which the system fails to operate according to its specifications.

In this paper, we aim to provide *cost-aware* and *failure-aware* provisioning policies to extend the capacity 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 [2]. 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 advantage 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 II, we present the hybrid Cloud system model used in this paper. Related work is described in Section III. Section IV presents the generic resource provisioning policy including brokering strategy, dispatch sequences, and scheduling algorithms. The resource provisioning in a hybrid Cloud system with two

978-0-7695-4749-7/12 $26.00 © 2012 IEEE
DOI 10.1109/HPCC.2012.49

resource providers is described in Section V. The performance evaluation of the proposed policies is discussed in Section VI. Conclusions and future work are presented in Section VII.

II. System Model

In this section, we briefly present the hybrid Cloud system model (Figure 1) used in this paper. This is based on the InterGrid middleware designed and implemented in Cloudbus research group[1] [7].

There are two different types of resource providers 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 the IGG, it determines which resource provider to use. An IGG consists of a scheduler that interacts 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.

In this paper, we consider a broad rang of high-performance applications including many different jobs requiring large number of resources over short periods of time. Each job (i.e., a request) could include several tasks and they might be sensitive to communication networks in terms of delay and bandwidth. Therefore, a request is assumed to be contiguous and must be served with resources from a single resource provider. Also, we assume requests are not deadline constrained.

III. 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.

In [21], the authors evaluate the feasibility of Amazon data storage service, S3, for scientific data-intensive applications. They provide recommendations that would better suit these applications. In particular, the authors mention that the monetary costs for accessing S3 associates infinite durability, high availability and fast access is high. By contrast, data-intensive applications often do not always need all of these features at once.

In [6], the authors determine the cost performance tradeoff of running an astronomy application with various work flow models on Amazon's S3 or Amazon's EC2. On a Cloud,

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

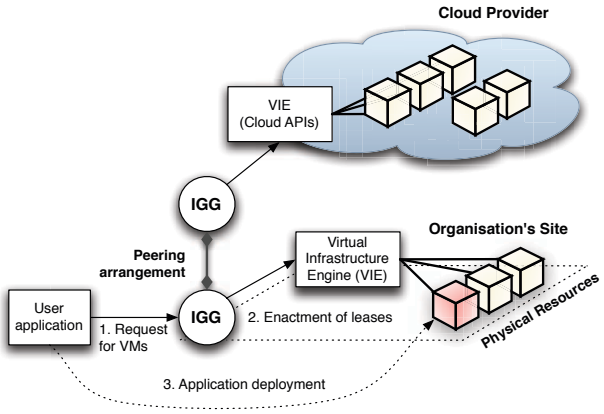


Fig. 1. The system architecture.

the cost of purchasing the resources may vary depending on computation, storage and communication overheads inherent in the application. This also impacts the performance of the application. The authors claim their application's computational cost outweighed the storage cost making the use of Cloud environment profitable without degrading the performance of their application.

Kondo *et al.* [16], provide a cost-benefit analysis between desktop grids and Amazon's elastic model. They try to answer several questions pertaining to these two platforms. One of the issues they address is the cost benefit of combining desktop grids with Cloud platform to solve large scale compute intensive applications. They conclude that hosting a desktop grid on a Cloud would be cheaper than running on stand alone desktop grids if bandwidth and storage requirements are less than 100Mbits and 10TB.

In [18], the authors proposed a model that elastically extends the physical site cluster with Cloud resources to adapt to the dynamic demands of the application. The central component of this model is an elastic site manager that handles resource provisioning. The authors provide extensive implementation and evaluation of their model.

Recently, Moschakis and Karatza [19] have evaluated the performance of applying Gang scheduling algorithms on Cloud. The authors address tasks that require frequent communication for which Gang scheduling algorithms are suitable. Their study is restricted to a single public Cloud which consists of a cluster of VMs on which parallel jobs are dispatched.

In the literature on resource provisioning in Cloud computing, most of the work has been on the implementation of cost performance benefit models rather than cost-aware performance models. In our queuing model, we take into account the total cost of dispatching and redirecting requests on both local cluster and public Clouds. In previous works, the cost of local resources is ignored and the model is simplified. Moreover, we consider the workload as the users' requests, which can be data or computing intensive applications.

Additionally, we incorporate resource failures into the

model, an important issue in any distributed system, thereby making our model a cost/failure-aware model. Moreover, our provisioning policies are based on non-observable queues that do not require any information about the input queue of either local cluster or public Clouds. Therefore, in contrast to the work by Assunção *et al.* [5], our proposed policies are independent of the scheduling algorithms.

IV. 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. In this section, we assume n providers and propose a generic solution for the three resource provisioning steps.

A. Adaptive Brokering Strategy

We formulate the problem of resource brokering similar to that of routing in the distributed parallel queues [2], [10]. 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 2.

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 IV-B). 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 a pay-as-you-go manner, the 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 [1].

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:

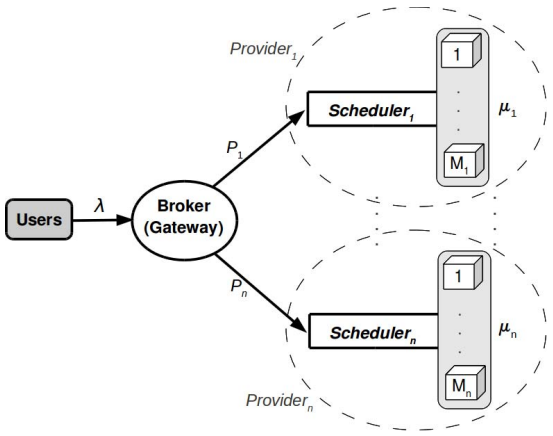


Fig. 2. Model of resource brokering for n providers.

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

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

Based on Figure 2, 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 [23] 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[2] [10]:

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

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:

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

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

where η_i can be calculated by the following equation:

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

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.

B. 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 (ADS) policy. To generate the deterministic sequence, we used the *Billiard* scheme [11], 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 [11], the authors proposed a method to generate the billiard sequence as follows:

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

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 [2]. 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.

C. 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*, *aggressive*, and *selective* backfilling [25]. 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.

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* [3] 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.

V. Case Study: Hybrid Cloud With Two Providers

In this section, we adopt the results of Section IV 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 Section II).

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 [10]. 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).

A. 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 IV, *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 the local cluster is failure-prone, we must consider the availability and unavailability intervals of each resource to find 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.* [14] to find the mean and coefficient of variance of completion time for W time units of work over M transient processors, as follows:

$$\overline{f} = \frac{W}{\overline{b}} = \frac{W}{M}\frac{(t_a + t_u)}{t_a} \tag{8}$$

$$\frac{\sigma_f}{\overline{f}} = \frac{\sqrt{\sigma_b^2}}{\sqrt{\overline{b}W}} \tag{9}$$

where

$$\overline{b} = \frac{t_a}{(t_a + t_u)}M \tag{10}$$

$$\sigma_b^2 = \frac{\sigma_a^2 t_u^2 + \sigma_u^2 t_a^2}{(t_a + t_u)^3}M \tag{11}$$

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 VI-A).

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:

$$\mu_s = \left(\frac{\overline{W}}{M_s \cdot \tau_s}\frac{t_a + t_u}{t_a} + L_s\right)^{-1} \tag{12}$$

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 (9). This makes our brokering strategy *failure-aware* and consequently adaptive to the system's failure pattern [13].

B. Runtime Model for Public Cloud

Although resource failures are inevitable, but public Cloud providers employ efficient and expensive mechanisms to manage resource failures. These mechanisms are mainly based on redundancy. Hence, public Cloud providers are usually able to provide *highly reliable* services to their customers [1]. 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:

$$\mu_c = \left(\frac{\overline{W}}{M_c \cdot \tau_c} + L_c\right)^{-1} \tag{13}$$

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 [20]. 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 [1]).

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

VI. Performance Evaluation

In order to evaluate the performance of the proposed policies, we implemented a discrete event simulator using CloudSim [4]. 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) [9] and the bounded slowdown [8]. The AWRT for N given requests is defined by the following equation:

$$AWRT = \frac{\sum_{j=1}^{N} d_j \cdot v_j \cdot (ct_j - st_j)}{\sum_{j=1}^{N} d_j \cdot v_j} \tag{14}$$

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:

$$Slowdown = \frac{1}{N}\sum_{j=1}^{N}\frac{w_j + max(d_j, bound)}{max(d_j, bound)} \tag{15}$$

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 [8].

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 failure-prone local cluster.

A. Workload Model

The workload model for evaluation scenarios is obtained from the Grid Workload Archive [12]. We used the *parallel job* model of the DAS-2 system which is a multi-cluster Grid [17]. Based on the workload characterization, the inter-arrival time, request size, and request duration follow Weibull, two-stage

TABLE I
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

TABLE II
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

Loguniform and Lognormal distributions, respectively. These distributions with their parameters are listed in Table I. 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:

$$\overline{V} = P_{one} + 2^{\lceil r \rceil}(P_{pow2}) + 2^r\left(1 - (P_{one} + P_{pow2})\right) \quad (16)$$

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

$$r = \frac{ql + m + (1-q)h}{2} \quad (17)$$

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

$$\overline{D} = e^{\theta + \frac{\sigma^2}{2}} \quad (18)$$

B. 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. In our experiments, the results of simulations are accurate within a confidence level of 95%.

The number of resources in the local cluster and Cloud is equal to $M_s = M_c = 64$ with homogeneous computing speed[3] (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[4] on a 10 Mbps network connection.

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

[4]This is the maximum amount of data for a real scientific workflow application [22].

The failure trace for the experiments is obtained from the Failure Trace Archive [15]. 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 failure events per node. The parameters for the failure model of Grid'5000 are listed in Table II (see [15] 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 I. 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 I.

To compute the cost of using resources from a 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:

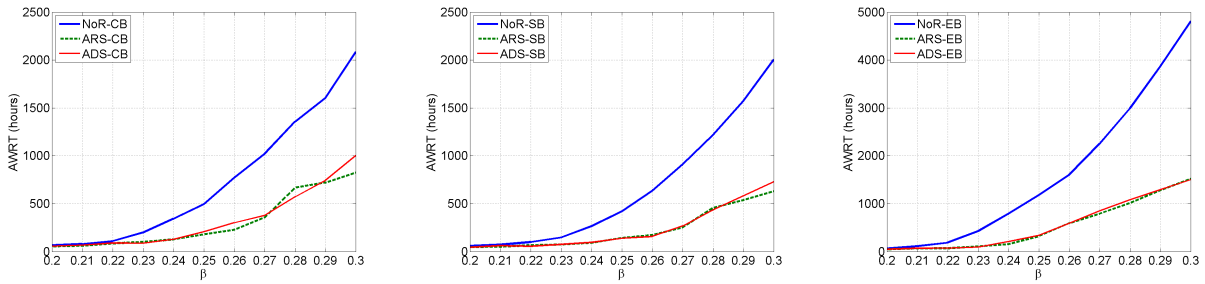
$$C_{EC2} = \left(H_c + N \cdot P_c \cdot H_u\right) C_p + \left(N \cdot P_c \cdot B_{in}\right) C_x \quad (19)$$

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 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 [20]. 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 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 EC2 is also considered as C_x which is 0.1 USD per GB [5]. 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 [1].

C. Results and discussions

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.

[5]All prices obtained at time of writing this paper.



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

Fig. 3. AWRT versus arrival rate ($\theta = 3.0$).

The simulation results for AWRT versus arrival rate are depicted in Figure 3 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 scheduler. Therefore, the ADS achieves almost the same performance as the ARS for all the scheduling algorithms.

Figure 4 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.

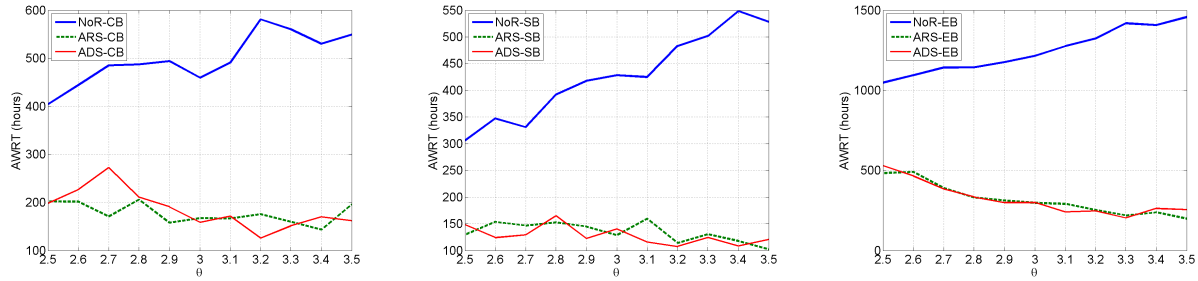
Figure 5 expresses slowdown of requests versus arrival rate for different provisioning policies with the same configuration as previous experiments. Based on these figures, use of adaptive brokering strategies decreases the request slowdown by 4 times for conservative and selective backfilling and 10.9 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 6, 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 [2] 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.

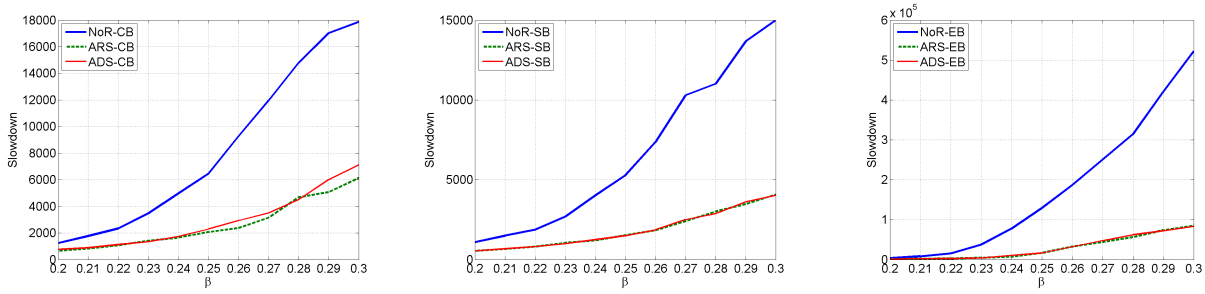
To analyze how the proposed policies utilize the Cloud resources, we calculated the amount of Cloud usage and its associated cost per month using Equation (19). We observed that the amount of Cloud usage (H_c in Equation (19)) is the same for all scheduling algorithms. However, as we illustrated before in this subsection, 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).

Table III lists the cost and the performance improvement of all proposed polices with respect to the case of using only the local failure-prone cluster. In each row the values are presented for different request duration while the input load is moderate (i.e. $\beta = 0.25$). 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 III, 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



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

Fig. 4. AWRT versus request duration ($\beta = 0.25$).



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

Fig. 5. Slowdown versus arrival rate ($\theta = 3.0$).

policy in terms of the cost at EC2 per month.

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 provisioning. 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.

VII. Conclusions

We considered the problem of cloud computing resource provisioning to extend the computing capacity of failure-prone local cluster. We presented a generic resource provisioning model based on the stochastic analysis of routing in distributed parallel queues where the 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.

Acknowledgments

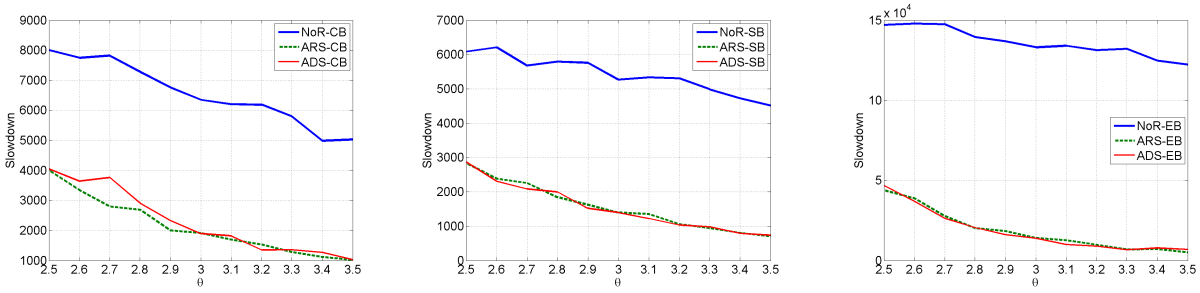
The authors would like to thank Jonatha Anselmi, Rodrigo N. Calheiros, and Mohsen Amini for useful discussions.

References

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

[2] J. Anselmi and 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.

[3] M.-S. Bouguerra, T. Gautier, D. Trystram, and J.-M. Vincent. A flexible checkpoint/restart model in distributed systems. In *9th International Conference on Parallel Processing and Applied Mathematics*, volume 6067 of *LNCS*, pages 206–215. Springer, 2010.



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

Fig. 6. Slowdown versus request duration ($\beta = 0.25$).

TABLE III
COST AND PERFORMANCE IMPROVEMENT OF PROVISIONING POLICES FOR DIFFERENT REQUEST DURATIONS AND MODERATE INPUT LOAD ($\beta = 0.25$).

Request	**EC2 cost/mth (USD)**		**Conservative**				**Selective**				**EASY**			
	ARS	ADS	ARS		ADS		ARS		ADS		ARS		ADS	
			AWRT	*Sld*	*AWRT*	*Sld*	*AWRT*	*Sld*	*AWRT*	*Sld*	*AWRT*	*Sld*	*AWRT*	*Sld*
Short ($\theta = 2.5$)	728.40	724.20	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 ($\theta = 3.0$)	1193.60	1191.60	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 ($\theta = 3.4$)	1434.40	1423.80	3.69	4.46	3.11	3.93	4.65	5.88	5.05	5.97	5.90	17.61	5.36	15.97

[4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and 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):23–50, 2011.

[5] M. D. de Assunção, A. di Costanzo, and R. Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *18th International Symposium on High Performance Parallel and Distributed Computing (HPDC 2009)*, pages 141–150, Garching, Germany, June 2009.

[6] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the Cloud: The montage example. In *19th ACM/IEEE InternationalConference on Supercomputing (SC 2008)*, pages 1–12, Piscataway, NJ, USA, 2008.

[7] 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.

[8] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *3rd Job Scheduling Strategies for Parallel Processing (IPPS'97)*, pages 1–34, London, UK, 1997.

[9] C. Grimme, J. Lepping, and A. Papaspyrou. Prospects of collaboration between compute providers by means of job interchange. In *13th Job Scheduling Strategies for Parallel Processing*, volume 4942 of *Lecture Notes in Computer Science*, pages 132–151, Berlin / Heidelberg, April 2008.

[10] X. Guo, Y. Lu, and M. S. Squillante. Optimal probabilistic routing in distributed parallel queues. *SIGMETRICS Perform. Eval. Rev.*, 32(2):53–54, 2004.

[11] A. Hordijk and D. van der Laan. Periodic routing to parallel queues and billiard sequences. *Mathematical Methods of Operations Research*, 59:173–192, 2004.

[12] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema. The Grid Workloads Archive. *Future Generation Computer Systems*, 24(7):672–686, 2008.

[13] B. Javadi, D. Kondo, J.-M. Vincent, and D. P. Anderson. Mining for statistical models of availability in large-scale distributed systems: An empirical study of SETI@home. *IEEE Transaction on Parallel and Distributed Systems*, 22(11):1896 –1903, nov. 2011.

[14] L. Kleinrock and W. Korfhage. Collecting unused processing capacity: An analysis of transient distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(5):535–546, 1993.

[15] D. Kondo, B. Javadi, A. Iosup, and D. H. J. Epema. The Failure Trace Archive: Enabling comparative analysis of failures in diverse distributed systems. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID)*, pages 398–407, 2010.

[16] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson. Cost-benefit analysis of Cloud computing versus desktop grids. In *23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009)*, pages 1–12, Rome, Italy, 2009.

[17] H. Li, D. Groep, and L. Wolters. Workload characteristics of a multi-cluster supercomputer. In *10th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 176–193, New York, USA, 2004.

[18] P. Marshall, K. Keahey, and T. Freeman. Elastic site: Using clouds to elastically extend site resources. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID)*, pages 43–52, 2010.

[19] I. A. Moschakis and H. D. Karatza. Evaluation of gang scheduling performance and cost in a cloud computing system. *Journal of Supercomputing*, Sep. 2010.

[20] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. A performance analysis of EC2 cloud computing services for scientific computing. In *1st International Conference on Cloud Computing (CloudComp 2009)*, pages 115–131, Munich, Germany, 2009.

[21] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon S3 for science Grids: a viable solution? In *1st International Workshop on Data-aware Distributed Computing (DADC'08) in conjunction with HPDC 2008*, pages 55–64, New York, NY, USA, 2008.

[22] S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. E. Dobson, and K. Chiu. A grid workflow environment for brain imaging analysis on distributed systems. *Concurrency and Computation: Practice and Experience*, 21(16):2118–2139, 2009.

[23] S. M. Ross. *Stochastic Processes*. second edition. John Wiley and Sons, 1997.

[24] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14 –22, Sep. 2009.

[25] 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.