Cost-effective Provisioning and Scheduling of Deadline-constrained Applications in Hybrid Clouds

Rodrigo N. Calheiros and Rajkumar Buyya

Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia
{rnc, rbuyya}@unimelb.edu.au

Abstract. In order to meet distributed application deadlines, Resource Management Systems (RMSs) have to utilize additional resources from public Cloud providers when in-house resources cannot cope with the demand of the applications. As a means to enable this feature, called Cloud Bursting, the RMS has to be able to determine when, how many, and for how long such resources are required and provision them dynamically. The RMS has also to determine which tasks will be executed on them and in which order they will be submitted (scheduling). Current approaches for dynamic provisioning of Cloud resources operate at a per-job level, ignoring characteristics of the whole organization workload, which leads to inefficient utilization of Cloud resources. This paper presents an architecture for coordinated dynamic provisioning and scheduling that is able to cost-effectively complete applications within their deadlines by considering the whole organization workload at individual tasks level when making decisions and an accounting mechanism to determine the share of the cost of utilization of public Cloud resources to be assigned to each user. Experimental results show that the proposed strategy can reduce the total utilization of public Cloud services by up to 20% without any impact in the capacity of meeting application deadlines.

1 Introduction

Advances in Cloud computing made available a virtually infinite amount of resources hosted by public Cloud providers that charge for resource utilization in a pay-per-use model [1]. Public Cloud infrastructures can be combined with existing in-house resources from organizations in order to accelerate the execution of their distributed applications. This technique is called *Cloud bursting*, and the environment comprising such combined resources is termed *Hybrid Cloud*.

When Cloud bursting is applied, the Resource Management System (RMS) coordinating the access to the resources has to determine when, how many, and for how long such resources are required and provision them dynamically. The RMS has also to determine which tasks will be executed on each resource and in which order (*scheduling*). A common approach to manage such access is to

assign an allocation time where a user has exclusive access to a number of resources. More sophisticated resource managers such as Oracle (former Sun) Grid Engine [2] and Aneka [3] operate in a different mode where tasks that compose the application are queued and executed whenever there are free resources in the infrastructure. Priority of tasks are periodically recalculated, what enables enforcement of organization-defined policies about access rights and Quality of Service (QoS) in the form of *deadlines* for application completion.

Even though several research projects focus on each of these steps individually (see Section 2), there is a lack of research in approaches that combine both activities in order to optimize resource utilization, minimize cost during provisioning, decrease execution time of applications, and meet deadlines. Moreover, most approaches for dynamic provisioning operate in a per-job level, and thus they are inefficient because they fail in consider that other tasks could utilize idle cycles of Cloud resources. The latter aspect is especially relevant in the context of typical Infrastructure as a Service (IaaS) providers, which charge users in specific time intervals (typically one hour) even if resources are utilized for just a fraction of the period.

To counter such lack of solutions for cost-effective dynamic provisioning and scheduling in hybrid Clouds, we present a coordinated dynamic provisioning and scheduling approach that is able to cost-effectively complete applications within their deadlines by considering the whole organization workload at individual tasks level when making decisions. The approach also contains an advanced accounting mechanism to determine the share of the cost of utilization of public Cloud resources to be assigned to each user.

The key contributions of this paper are: (i) It proposes an architecture to enable coordinated dynamic provisioning of public Cloud resources and scheduling of deadline-constrained applications; (ii) It proposes a strategy for combined dynamic provisioning and scheduling of tasks; and (iii) It proposes a novel approach for billing users for the utilization of public Cloud resources. Experimental results show that the proposed strategy can reduce the total utilization of public Cloud services by up to 20% without any impact on the capacity of meeting application deadlines.

2 Related Work

The most of the existing scheduling policies for Clusters, Grids [4–10], and hybrid Clouds [11–13] either operate with user specification of allocation slots for utilization of resources or make decisions for a single job without considering jobs already queued. In the latter approach, decisions that optimize one job may cause delays to other jobs or, when Cloud resources are provisioned to complement local resources, may lead to underutilization of the extra resources. In the former approach, users are responsible for ensuring that the job can be executed within the time slot. However, users typically overestimate their jobs' needs, what leads to inefficiencies in the scheduling process. Therefore, we apply a request model where users do not reserve resources during a time interval for job

execution. Instead, users submit jobs and specify their deadlines (if any), and the scheduler submits tasks for execution on resources.

The above model is also adopted by the Sun Grid Engine (SGE) [2] and the systems derived from it. Such systems offer a scheduling policy for distributed jobs that allows priority to be assigned to users or groups. It also contains a model of deadline for job execution. However, in such a system deadline is defined in terms of *start time* of the job, whereas our model considers the *completion time* of the job. UniCloud[1] is a software that allows Univa Grid Engine (a derivative from SGE) to provision resources from public Clouds. However, provision of public Cloud resources is manually managed by system administrators.

Lee and Zomaya [14] propose an algorithm for scheduling of Bag of Tasks applications on hybrid Grids and Clouds. This algorithm assigns tasks to Cloud resources only for rescheduling purposes, whereas our approach deploys Cloud resources to meet tight deadlines. Van den Bossche *et al.* [15] propose a heuristic for cost-efficient scheduling of applications in hybrid Clouds. In such work, the application model is similar to the application model addressed by our research. However, their approach makes decision of whether using in-house resources or Cloud resources at job level (i.e., all the tasks that compose the job either run in-house or run on the Cloud) without reutilization of Cloud resources. Our approach, on the other hand, schedules at task level. This has the advantage of enabling a better utilization of Cloud resources by running tasks from other jobs if the billing interval is not over and the job that requested the Cloud resources finished.

Dynamic provisioning of Cloud resources has been explored with different purposes. Vázquez *et al.* [16] present an architecture for dynamic provisioning of Cloud resources to extend the capacity of a Grid in response to events in the RMS. However, the paper does not present any method to determine when the Cloud resources should be deployed or decommissioned. Therefore, the architecture presented in this paper complement such previous work.

Mateescu *et al.* [17] propose a hybrid Cloud environment for HPC applications. Such a system manages requests at single task level. Therefore, deadlines are determined for individual tasks, not for the whole job. It provisions resources from public Clouds to increase probability that tasks start their execution within the start deadline, oppositely to a completion deadline model used in this paper.

Mao *et al.* [18] proposes an auto-scaling mechanism for provisioning resources to jobs in order to meet deadlines. The approach, however, only considers the provisioning problem, while we adopt an integrated provisioning and scheduling mechanism to meet application deadlines.

In our previous work [19], we investigated dynamic provisioning techniques in hybrid Clouds and applied it in the Aneka Cloud platform. However, the approach is applied for individual jobs only and is not integrated with the scheduler; therefore it is not cost-effective in the presence of multiple simultaneous jobs with deadlines.

[1] `http://www.univa.com/products/unicloud`

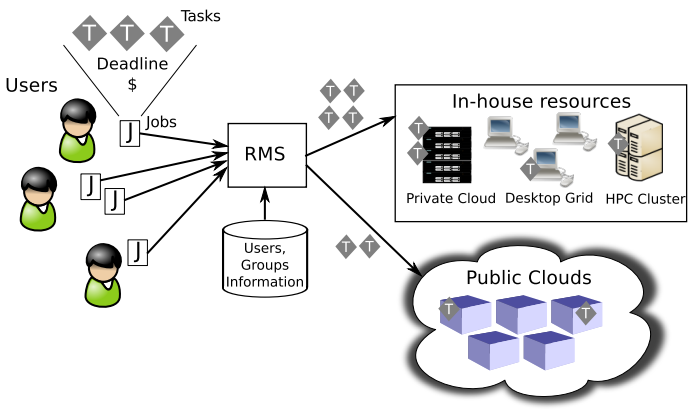

Fig. 1. System and application models assumed in this paper. Jobs are composed of independent tasks, and they can also contain deadline and budget specification. The Resource Management System (RMS) deploys Hybrid Cloud resources to execute tasks and meet deadlines. Decisions are made by the RMS with the support of information about users, groups they belong to, and their access rights.

3 System and Application Models

The system model assumed in this paper is depicted in Figure 1. The central component of the model is the Resource Management System (RMS) that manages a number of local resources (private Cloud). The specific nature of the private Cloud is irrelevant from the system perspective. It may be composed of desktop Grids, a HPC Cluster, or a virtualized data center. Examples of RMSs that follow such a model are Oracle (former Sun) Grid Engine [2] and its derivatives and Manjrasoft Aneka [3].

The RMS has access to one or more public Cloud providers that lease resources in a pay-per use manner. Resources are leased by the RMS via a specific provisioning request sent to the Cloud provider. In such a request, the RMS specifies characteristics of the resources and number of resources required. When Cloud resources are no longer required by the RMS, a decommission request is sent to the public Cloud provider, which releases the resources. Use of Cloud resources is charged in time intervals whose durations are defined by Cloud providers (typically, one hour). Use of a fraction of the time interval incurs in the payment of the whole interval.

The RMS is accessed by users who want to submit loosely-coupled distributed applications in the resources managed by the system. The user request (job) contains (i) description of each task that composes the job, including required files, estimated runtime; and (ii) optional QoS attributes in the form of deadline for job completion and budget to be spent to meet the deadline.

The proposed model does not require that tasks have homogeneous execution time. Therefore, it suits both Bag of Tasks and Parameter Sweep applications.

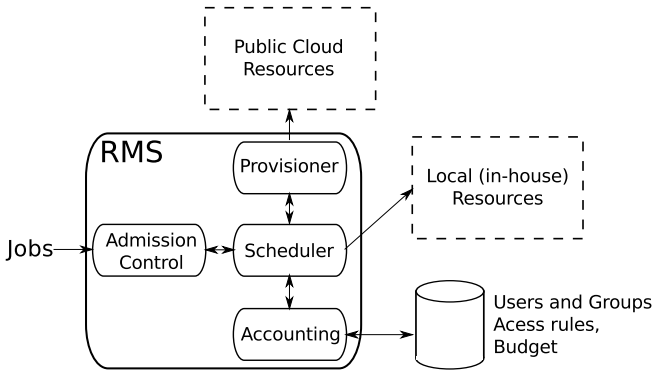

Fig. 2. Proposed Resource Management System architecture for integrated dynamic provisioning and scheduling of applications in hybrid Clouds.

Support for dependencies among tasks that constitute a job (which enables support for Workflow applications) is the subject of future work. Furthermore, we assume that files required by tasks are stored in the in-house infrastructure. Therefore, file transferring is required only in the case of Cloud execution.

Tasks from different users compete for resources, and the RMS determines which tasks execute in a given moment and where. However, this has to be done without causing starvation to any job in the waiting queue (i.e., the RMS has to guarantee that each job will eventually complete). Furthermore, the organization can enforce policies about access rights of users and groups, which have to be taken into account by the RMS.

When the RMS detects that one or more jobs are risking missing their deadlines, provisioning policies are applied so that resources are acquired and deployed to speed up such jobs. However, because charge for public Cloud resource utilization is made by a time slot that can be bigger than the runtime of the tasks of the job that required it, the RMS has to apply a reuse policy in order to improve the utilization of the public Cloud resources.

4 Proposed Architecture

Our proposed RMS architecture is depicted in Figure 2. Requests for job execution are received by the *Admission Control* component. Accepted requests are received by the *Scheduler* component. Based on information about job queues, jobs' deadlines, and amount of available resources, requests for extra resources are sent from the Scheduler to the *Provisioner*. The Provisioner is responsible for acquiring resources from public Clouds and making them available to the Scheduler. Finally, the *Accounting* module interacts with the Scheduler to determine whether users have credit and authorization to request and use resources from public Clouds, and also to keep track of utilization of external resources so groups and users can be properly charged for public Cloud utilization.

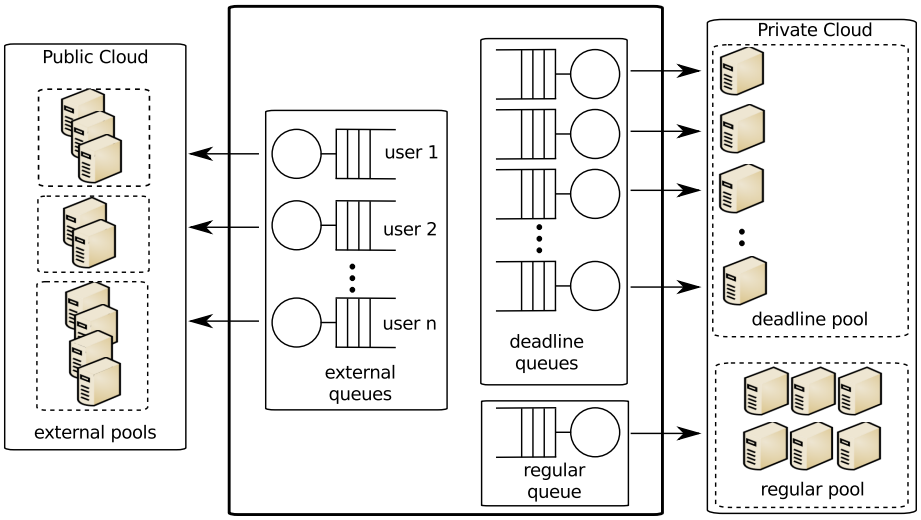

Fig. 3. Organization of resource pools and scheduling queues.

The Admission Control accepts all the requests that do not have a deadline constraint. For requests with deadlines, it makes the decision whether the job can be accepted and completed within the deadline or the job must be rejected because it is unfeasible. To determine whether a request can be accepted, the Scheduler module is queried by the Admission Control module. The Scheduler than, considering user estimation, available resources, Cloud resources in use, and user access rights, replies to the Admission Control whether the user has permission and credit to run the job and whether the deadline is feasible or not. The Scheduler's reply is used as the final decision about job acceptance.

4.1 Scheduler

Jobs that are accepted by the Admission Control are received by the Scheduler module, which makes decisions based on a number of factors such as the pool to which the idle resources belongs to and job priority and ownership.

In order to prevent starvation of regular jobs, a minimum amount of resources to be made available for regular tasks can be defined. These resources compose the *regular pool* and its access is coordinated via a *regular queue*. The rest of the local machines belong to the *deadline pool*, whose accesses are coordinated via *deadline queues*. Finally, dynamically provisioned machines belong to *external pools* and are coordinated by *external queues*. Figure 3 depicts the organization of the resource pools and queues in the Scheduler.

Tasks that compose submitted jobs are forwarded either to the *regular* queue or to one of the *deadline* queues (there is one of such queues for each resource that belongs to the deadline pool). They respectively store tasks without deadline-constraints and tasks with such constraints. Tasks on each queue are rearranged

every time a new job is received by the Scheduler and every time a task completes. A third set of queues, *external* is also present in the Scheduler. There is one of such queues for each user and it contains tasks that belong to jobs that require dynamic provisioning to complete before the deadline. Tasks on this queue execute preferentially in dynamically provisioned resources, as detailed later in this section.

Algorithm 1 details the procedure for building the regular queue. This procedure runs every time a new job is received and every time a new resource is added to this pool. The total time of each resource used by jobs from a group is summed up to give the total work w_g of group g (Lines 2 to 5). Groups are sorted in ascending order of w_i (Line 6), and each group receives a share of resources N_i that respects the amount of resources assigned to each group defined in the Scheduler (Line 8). The value N_g is the number of tasks from the group g that go to the top of the queue. N_g tasks from the group with the lowest w_i go to the top of the queue, followed by N_h tasks from the group with the second lowest w_i and so on, until all the shares are defined. The rest of the tasks are put in the end of the queue in arrival order (Line 11).

In the case of the deadline pool, whenever a new job is received, tasks are scheduled to different resource queues following a policy such as Round Robin, Worst Fit, Best Fit, and HEFT [20]. We do not apply backfilling techniques to prioritize tasks with closer deadlines because it may motivate users to make late submission of jobs or to overestimate execution time of tasks (both situations that would increase priority of their jobs over others).

Dispatching of tasks for execution depends on the pool that the idle resource belongs to. When a resource from the regular pool becomes idle, the task on top of the regular queue is dispatched for execution in such resource. If the regular queue is empty, the waiting task from deadline queues with the smallest *lag time* (which we define as the difference between the time to the deadline and the estimated execution time) is removed from its queue and dispatched for execution. Finally, if the deadline queue is also empty, the first task on the external queue is dispatched for execution.

When a resource from the deadline pool becomes idle, the next task on its queue is dispatched. If the queue is empty, the task from other queues with the smallest lag time is removed from its original queue and dispatched. If the deadline queue is empty, the first task in the external queue is dispatched or, if the queue is empty, the first task in the regular queue is dispatched.

Whenever a resource from the external queue becomes available, the first task on the external queue that belongs to the user that required the resource is dispatched to the resource. If there is no such a task, a task from the user is sought in the deadline queue. The first task from the user whose estimated execution time is smaller than the time left before the end of the resource's billing period is dispatched. If no task from the user meets this condition, the first task from the user in the regular queue is dispatched.

When the user that requested the resources does not have tasks to execute, the Scheduler applies the same procedure discussed in the previous paragraph

Algorithm 1: Regular scheduler queue build up procedure.

Data: res: number of resources in the regular pool.
Data: max_i: maximum number of resources allowed for the group i.
empty regular queue;
foreach *group* g_i **do**
    $w_i \leftarrow w_i / \sum_j w_j$, the proportional resource utilization by group g_i during the current time window;
    utilizationList $\leftarrow w_i$;
end
sort utilizationList in ascending order of w_i;
foreach w_i *in utilizationList* **do**
    $share_i \leftarrow min(max_i, \lceil (1 - w_i) * res \rceil)$;
    add $share_i$ tasks from group g_i to the regular queue;
end
add remaining tasks in FIFO order to the regular queue;

for tasks from the group that required the Cloud resources. If no other task from the same group is found, the procedure is applied for tasks from other groups.

4.2 Provisioner

The Provisioner makes decisions about utilization of public Cloud resources. It calculates the number of extra resources required to execute a job within its deadline and also decides if machines whose billing periods are finishing will be kept for another period or not. The required number of resources is defined at task level: tasks that belong to an accepted job that can run in the deadline pool before the deadline are scheduled locally. Tasks that cannot be completed on time are put in the external queue by the scheduler, and provisioning decision is made based only on such tasks. Currently, the provisioner assumes a single type of VM to be provisioned. This increases the chance of successful allocation of Cloud resources because it enables acquisition of "reserved" or "pre-paid" resources. Most IaaS offer such type of resource, which guarantees that, whenever resources are required, they will be available, as users paid for them upfront or via a premium plus discounted rates for utilization. Alternatively, the provisioner can register multiple providers, and use resources from another provider when the preferable one cannot supply the required resources.

When a virtual machine is reaching the end of its billing period, the Provisioner decides whether the resource should be kept for the next billing period or if it should be decommissioned. This decision is based on the states of external deadline queues. The simplest case is when the external resource is idle or it is running a regular task. It happens when the other queues are empty. In this case, the resource is decommissioned by the Provisioner. A regular task running on the resource is rescheduled in the regular queue. If the provisioned resource is executing a deadline or external task, the resource is kept for the next billing period to avoid risk of missing the job's deadline.

In the case that the resource is no more necessary for the user that originally requested it, and there are still external tasks in the queue, the resource is reassigned for the user that needs the resource (providing it has authorization and credit to use them). In this case, accounting responsibilities for the reassigned resource is also changed, as detailed next.

4.3 Accounting

When Cloud resources are deployed, users and/or groups have to be made accountable for the extra cost incurred by such resources. This is required for reducing the operational costs of the organization. Furthermore, even though accounting is made at user and group level, the system has to apply policies to keep utilization of such external resources as high as possible, so the investment in Cloud resources can be justifiable.

In order to achieve such goals, we propose the *Reassignable Ownership Policy* that operates as follows. Each Cloud resource is associated to an *owner*. Resource ownership is determined by the Provisioner. The resource owner is accountable for any period of idleness of the machine, as well all the period when it was running its tasks on the resource. However, during the period where deadline or external tasks from other users are executed, the corresponding period is assigned to task owners. Moreover, any time a regular task belonging to a user that is not the resource owner is running in Cloud resources, the corresponding period is excluded from the usage period of the owner.

The actual debt the user or group has with the organization corresponds to the fraction of the price per billing period that the user/group was made accountable for. The corresponding fraction of the resource cost is then charged by the accounting module. This enables users to amortize part of the cost related to use Cloud resources and also allows the whole organization to fully utilize Cloud provisioned resources.

5 Performance Evaluation

In this section, we present experiments aiming at evaluating the proposed integrated dynamic provisioning and scheduling technique and its impact in terms of QoS and overall cost of utilization of public Cloud infrastructures.

5.1 Experiment Setup and Workload

Experiments were conducted using the CloudSim toolkit [21] for discrete event simulation. The simulated hybrid Cloud is composed of a local infrastructure managed by a RMS and a public Cloud used for Cloud bursting purposes. The local infrastructure contains 100 virtual machines (VMs). Each machine has 4GB of RAM and a single core processor. The public Cloud accepts requests for up to 100 single core virtual machines from the RMS. Each VM in the public Cloud has the same capacity than the in-house VMs. We assume a negligible latency

for communication between the RMS and the in-house infrastructure, and 500 ms latency between the RMS and the public Cloud.

CloudSim applies a "relative" measurement of CPU power, defined as million instructions per second (MIPS), whereas tasks are described in millions of instructions (MIs). Therefore, tasks are defined in terms of how much CPU time is required for its execution assuming no time-sharing of resources. Throughout this section, we refer to this relative time to determine task characteristics.

The RMS is subject to a 24-hours long sequence of job submissions following an adapted version of the BoT workload model proposed by Iosup *et al.* [22], which was derived from the analysis of utilization traces of seven Grids worldwide. According to this workload model, the interarrival time of a BoT job in peak time follows a Weibull distribution with parameters (4.25, 7.86). However, for this experiment purposes, we assume that during the 24 hours period submission of jobs follows the peak time pattern. Furthermore, we varied the arrival rate of jobs by modifying the parameters of the Weibull distribution. This allowed us to evaluate the system performance subject to different load conditions: we run experiments using two different values for the scale of the distribution (first parameter of the distribution: 4.25 and 8.5) and three different values for the shape (second parameter of the distribution: 7.86, 3.93, 15.72).

The number of tasks of each job request is defined in the workload model as 2^x, where x follows a Weibull distribution with parameters (1.76, 2.11). We assume that tasks that compose a job are homogeneous regarding execution time. The runtime of tasks, as defined by the aforementioned workload model, is 2^x minutes, where x follows a normal distribution with average 2.73 and standard deviation 6.1.

Finally, Iosup's workload model does not contain a description on how to assign deadlines for the each job. Therefore, deadlines for each job were assigned following a method proposed by Garg *et al.* [23]. Such a method divides jobs in two urgency classes, namely low-urgency and high-urgency jobs. Jobs are assigned to each class uniformly according to a defined share. We evaluated two different shares of high-urgency jobs: 20% and 50%.

Deadlines of jobs on each class vary in the ratio deadline/runtime, as follows. High-urgency jobs have such a rate sampled from a uniform distribution with average 3 and standard deviation 1.4 (i.e, in average the deadline is 3 times of the estimated runtime), whereas low-urgency jobs have such a rate sampled from a uniform distribution with average 8 and standard deviation 3. The obtained value for deadline is counted from the moment the job is submitted for execution to the RMS. Finally, ownership of jobs was assigned to 10 groups, each one with 1 user, following a uniform distribution.

Because the proposed method operates with reservation of resources for composing the regular pool responding to execute regular tasks, and because regular tasks are not subject to QoS metrics, we ignore regular jobs and the regular resource pool for the purpose of these experiments. 24 hours-long workloads with different arrival rates generate according to the above method were submitted for execution in the simulated hybrid Cloud. Experiments for each combination

of arrival rates (six different combinations of shape and scale) and urgency (two different rates), which resulted in 12 different scenarios, were repeated 30 times.

For performing the scheduling of the tasks on the deadline queues, we applied the Heterogeneous Earliest Finish Time (HEFT) algorithm [20]. This is a well-known and efficient algorithm for scheduling of applications on heterogeneous environments. Its main advantages are its low complexity and high performance in terms of reducing application execution time.

The HEFT algorithm was used together with two different dynamic provisioning techniques. The first is a job-level provisioning similar to several previous works in the area. When this technique is applied, all the tasks that belong to the new job are removed from the deadline queues and moved to the external queue. Dynamically provisioned resources for the job are decommissioned when the job completes. This technique is labeled as *job-based* in the experiment results. The second technique is the integrated dynamic provisioning and scheduling operating at task-level proposed in this paper: only tasks whose deadline cannot be met with in-house resources are executed in the public Cloud. This technique is applied together with the proposed Reassignable Ownership Policy and is labeled as *Integrated* in the experiment results.

The two strategies of provisioning and scheduling are evaluated with each one of the workloads generated as discussed previously. We report for each combination provisioning-scheduling the amount of jobs whose deadline was missed and the total utilization of public Clouds in terms of number of hours of instances allocated (a metric we call VM-hours).

Finally, it worth noting that, to allow evaluation of a scenario with minimal influence of other types of limitations caused by specific policies, the admission control mechanism was modified to accept all the jobs generated by the workload.

5.2 Results and Discussion

Figure 4 presents results for utilization of public Clouds. The unit used is VM-hours, which we define as the sum of the wall clock time of each dynamically provisioned VM, from its creation to its destruction. Results show that our integrated provisioning approach was able to successfully reduce the utilization of public Cloud resources as a whole. Utilization of public Cloud resources was reduced to up to 20%. The smallest improvement generated by our integrated strategy was 5.2%, and the average reduction in public Cloud utilization was 10.24%. It represents a significant reduction in costs for organizations considering that our experiment simulated 1 day of resources utilization for 10 users. Because typical utilization scenarios are likely to be scaled to a bigger number of users for longer periods of time, the application of our approach can help organizations to significatively reduce their budget of Cloud bursting.

Paired t-tests on the Cloud utilization reported by different policies showed that task-level scheduling and provisioning caused 1% increase in the utilization of local resources (because of tasks that were kept locally instead of being sent to the Cloud). Because the total number of hours of the workload is the same, and the increased local load was smaller, we conclude that the significant reduction

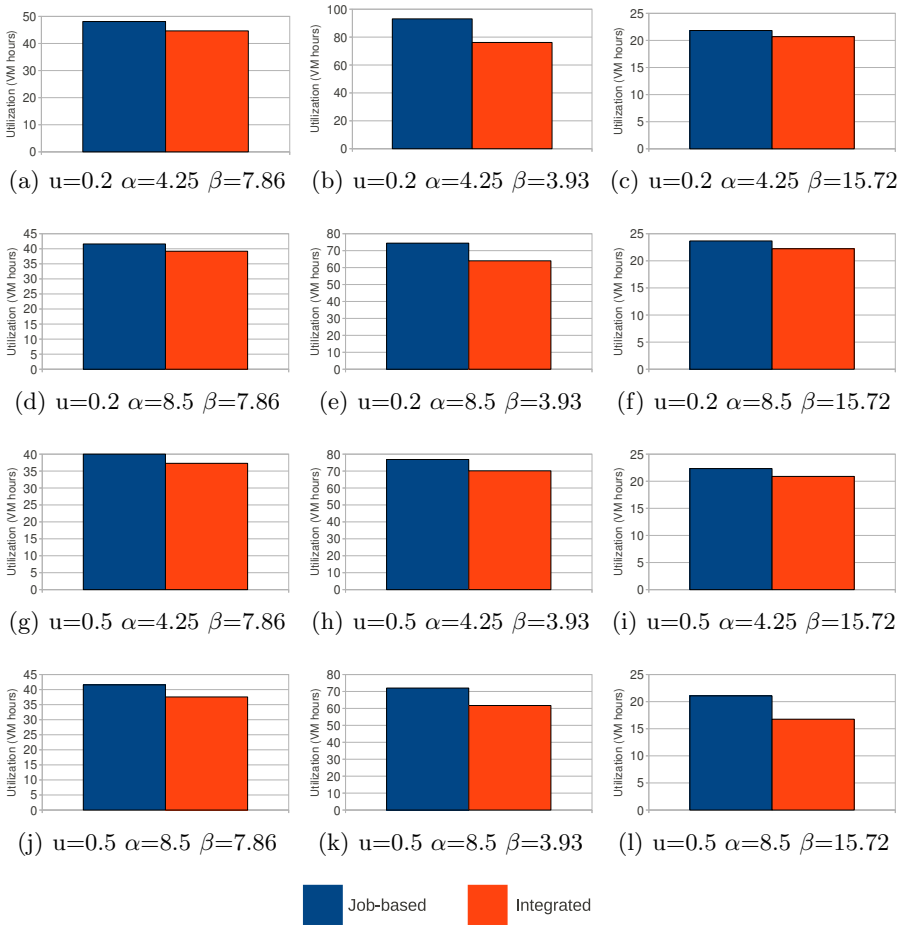

Fig. 4. Public Cloud utilization in VM-hours for a 1-day load workload for different simulation scenarios. Job-based: traditional provisioning techniques applied at job-level. Integrated: our integrated policy with task-level provisioning and reassignment of public resources. α and β denote the scale and shape of the job arrival distribution and u denotes the rate of urgent requests. The scheduling algorithm used in all scenarios is the HEFT algorithm. Note that different plots have different scales.

in the number of Cloud resources is caused by a more effective utilization of such resources. This reduction in Cloud utilization happened without any impact in the capacity of our mechanism in meeting job deadlines. In fact, all the policies were able to meet deadline of 100% of jobs by applying a provisioning strategy. This is an expected effect as, because the system is able to provisioning as many public Cloud resources as necessary, and no unfeasible jobs were submitted, deadlines could always be met with a sufficiently large amount of public Cloud resources.

6 Conclusions and Future Work

Cloud computing transformed the way that distributed applications are executed by making possible to complement in-house resources with pay-per use public Cloud resources. This makes possible for users to define a deadline for application execution and the budget to be spent, if necessary, for the deadline to be met.

In this paper, we presented an architecture that enables Resource Management Systems to support the aforementioned tasks. We describe the architecture, a combined provisioning and scheduling strategy, and an approach for billing users for utilization of Cloud resources that compensates resources reallocated to other users when the deadline application completes before the end of resource billing period. Simulation experiments show that our approach makes an efficient utilization of public Cloud resources and enables deadlines to be met with reduced expenditure with public Cloud resources by organizations.

As future research, we will investigate optimization strategies in order to enable better utilization of multicore resources, when they are available. We will also extend the algorithms to support workflows and other applications where the RMS has to consider dependencies between tasks during the scheduling.

References

1. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems **25**(6) (Jun. 2009) 599–616
2. Gentzsch, W.: Sun Grid Engine: towards creating a compute power grid. In: Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGrid'01), Brisbane, Australia (May 2001) 35–36
3. Vecchiola, C., Chu, X., Buyya, R.: Aneka: A software platform for .NET-based cloud computing. In Gentzsch, W., Grandinetti, L., Joubert, G., eds.: High Speed and Large Scale Scientific Computing. IOS Press, Amsterdam, The Netherlands (2009) 267–295
4. Feitelson, D.G.: Scheduling parallel jobs on clusters. In Buyya, R., ed.: High Performance Cluster Computing. Volume 1. Prentice-Hall, Upper Saddle River (1999)
5. Braun, T.D., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing **61**(6) (Jun. 2001) 810–837
6. Silva, D., Cirne, W., Brasileiro, F.: Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In Kosch, H., Böszörményi, L., Hellwagner, H., eds.: Euro-Par 2003 Parallel Processing. Volume 2790 of Lecture Notes in Computer Science., Springer (Aug. 2003) 169–180
7. Cooper, K., et al.: New grid scheduling and rescheduling methods in the GrADS project. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, USA (Apr. 2004)
8. Weng, C., Lu, X.: Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid. Future Generation Computer Systems **21**(2) (Feb. 2005) 271–280

9. Dong, F.: A taxonomy of task scheduling algorithms in the grid. Parallel Processing Letters **17**(4) (Dec. 2007) 439–454
10. Salehi, M.A., Javadi, B., Buyya, R.: Resource provisioning based on lease preemption in InterGrid. In: Proceedings of the 34th Australasian Computer Science Conference (ACSC'11), Perth, Australia (Jan. 2011)
11. Assunção, M.D., di Costanzo, A., Buyya, R.: Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In: Proceedings of the 18th International Symposium on High Performance Distributed Computing (HPDC'09), Munich, Germany (Jun. 2009) 141–150
12. Salehi, M., Buyya, R.: Adapting market-oriented scheduling policies for cloud computing. In Hsu, C.H., Yang, L., Park, J., Yeo, S.S., eds.: Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'10). Volume 6081 of Lecture Notes in Computer Science., Busan, South Korea, Springer (May 2010) 351–362
13. Moreno-Vozmediano, R., Montero, R.S., Llorente, I.M.: Multicloud deployment of computing clusters for loosely coupled MTC applications. IEEE Transactions on Parallel and Distributed Systems **22**(6) (Jun. 2011) 924–930
14. Lee, Y.C., Zomaya, A.: Rescheduling for reliable job completion with the support of clouds. Future Generation Computer Systems **26**(8) (Oct. 2010) 1192–1199
15. den Bossche, R.V., Vanmechelen, K., Broeckhove, J.: Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom'11), Athens, Greece (Dec. 2011) 320–327
16. Vázquez, C., Huedo, E., Montero, R.S., Llorente, I.M.: Dynamic provision of computing resources from grid infrastructures and cloud providers. In: Proceedings of the Workshops at the Grid and Pervasive Computing Conference, Geneva, Switzerland (May 2009) 113–120
17. Mateescu, G., Gentzsch, W., Ribbens, C.J.: Hybrid computing–where HPC meets grid and cloud computing. Future Generation Computer Systems **7**(5) (May 2011) 440–453
18. Mao, M., Li, J., Humphrey, M.: Cloud auto-scaling with deadline and budget constraints. In: Proceedings of the 11th International Conference on Grid Computing (GRID'10), Brussels, Belgium (Oct. 2010) 41–48
19. Calheiros, R.N., Vecchiola, C., Karunamoorthy, D., Buyya, R.: The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid clouds. Future Generation Computer Systems **28**(6) (Jun. 2012) 861–870
20. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems **13**(3) (Mar. 2002) 260–274
21. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: 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
22. Iosup, A., Sonmez, O., Anoep, S., Epema, D.: The performance of bags-of-tasks in large-scale distributed systems. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC'08), Boston, USA (Jun. 2008) 97–108
23. Garg, S.K., Yeo, C.S., Anandasivam, A., Buyya, R.: Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers. Journal of Parallel and Distributed Computing **71**(6) (Jun. 2011) 732–749