CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
*Concurrency Computat.: Pract. Exper.* (2013)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3004

# Resource provisioning based on preempting virtual machines in distributed systems

Mohsen Amini Salehi [1,*,†], Bahman Javadi [2] and Rajkumar Buyya [1]

[1] *Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia*
[2] *School of Computing, Engineering and Mathematics, University of Western Sydney, Australia*

SUMMARY

Resource provisioning is one of the main challenges in large-scale distributed systems such as federated Grids. Recently, many resource management systems in these environments have started to use the lease abstraction and virtual machines (VMs) for resource provisioning. In the large-scale distributed systems, resource providers serve requests from external users along with their own local users. The problem arises when there is not sufficient resources for local users, who have higher priority than external ones, and need resources urgently. This problem could be solved by preempting VM-based leases from external users and allocating them to the local ones. However, preempting VM-based leases entails side effects in terms of overhead time as well as increasing makespan of external requests. In this paper, we model the overhead of preempting VMs. Then, to reduce the impact of these side effects, we propose and compare several policies that determine the proper set of lease(s) for preemption. We evaluate the proposed policies through simulation as well as real experimentation in the context of InterGrid under different working conditions. Evaluation results demonstrate that the proposed preemption policies serve up to 72% more local requests without increasing the rejection ratio of external requests. Copyright © 2013 John Wiley & Sons, Ltd.

Received 19 October 2012; Revised 5 January 2013; Accepted 5 January 2013

KEY WORDS: lease preemption; virtualization; suspend; migrate; resource sharing; preemption policy

## 1. INTRODUCTION

Managing and providing computational resources for user applications is one of the challenges in the high-performance computing community. Resource-sharing environments enable sharing, selection, and aggregation of resources across several resource providers (RP), also known as sites, that are connected through high bandwidth network connections. In a resource-sharing environment, computational resources in each RP are shared between external users as well as local users of the RP. Recently, virtual machine (VM) technology has been employed for resource provisioning in many resource-sharing environments [1–3]. Nowadays, heavy computational requirements, mostly from scientific communities, are supplied by these resource-sharing environments, such as Grid'5000 [4] in France and DAS-2 [5] in the Netherlands.

InterGrid [6] is an example of resource-sharing environments that provides an architecture and policies for sharing resources among Grids. In InterGrid, resource provisioning and sharing is based on the *lease* abstraction [7]. A lease is an agreement between an RP and a resource consumer whereby the provider agrees to allocate resources to the consumer according to the lease terms presented by the consumer [3, 8]. VM technology has been used to implement lease-based resource provisioning [3]. The capabilities of VMs in getting suspended, resumed, stopped, or even migrated

*Correspondence to: Mohsen Amini Salehi, CLOUDS Lab, Department of Computer Science and Software Engineering, The University of Melbourne, Australia.

†E-mail: mohsena@csse.unimelb.edu.au

Copyright © 2013 John Wiley & Sons, Ltd.

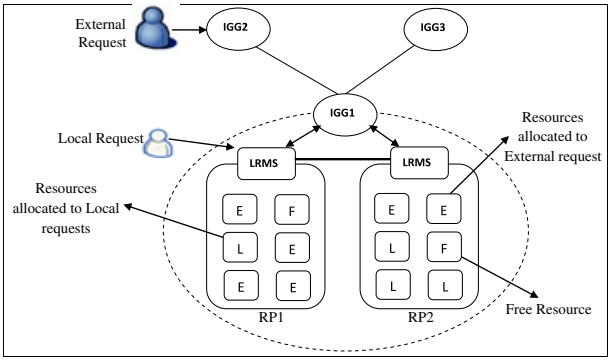


Figure 1. A scenario that shows the contention between local and external requests. IGG, InterGrid Gateway; LRMS, local resource manager system; RP, resource provider.

(when there is enough bandwidth) have been extensively studied and have shown to be useful in resource provisioning without major utilization loss [8–10]. InterGrid makes one lease for each user VM request.

As illustrated in Figure 1, resource provisioning in InterGrid is performed for different types of users, namely, local users and external users. Local users (hereafter termed local requests) refer to users who ask their local RP for resources. The provisioning rights over the resources from several RPs inside a Grid are delegated to the InterGrid Gateway (IGG). IGGs coordinate resource allocation for external requests through predefined arrangements between Grids [6]. External users (hereafter termed external requests) are those users who send their requests to the IGG to have access to larger amount of resources.

Typically, for an RP, local requests have more priority than external ones [9]. In other words, the organization that owns the resources would like to ensure that its community has priority access to the resources. In this circumstance, external requests are welcome to use resources if they are available. Nonetheless, external requests should not delay the execution of local requests. This scenario leads to resource contention between local and external requests to access resources.

In this paper, we resolve the resource contention between local and external requests in the InterGrid by preempting external leases in favor of local requests. However, preempting VM-based leases is not free of cost and involves various side effects. In this paper, we also deal with these side effects and propose policies to reduce their impact.

The rest of this paper is organized as follows: In Section 2, details of the problem we investigate in this paper is discussed; then, in Section 3, related research works are introduced. We explain the architectural framework including the InterGrid architecture and its extension for our proposed solution in Section 4. The model proposed for VM preemption time overhead is discussed in Section 5. Investigated policies are described in Section 6, and detailed evaluations are described in Section 7. The implementation of the investigated policies is discussed in Section 8. Finally, conclusion and future works are provided in Section 9.

## 2. PROBLEM STATEMENT

The main problem we are dealing with is resource procurement for local requests when existing resources have been allocated to external ones and the rest of resources are not adequate to serve the local requests. In this situation, one solution is to preempt external leases and allocate the resources to local requests [11]. However, preempting VM-based leases involves two main side effects.

The first side effect is the time overhead imposed on the system for preempting VM-based leases. The overhead varies based on the type of operation performed on the preempted VMs. For instance, if a VM is suspended after preemption, then the imposed overhead is lower than situation that it is migrated to another provider. The imposed overhead of preempting VMs can affect the resource utilization of an RP. This impact is remarkable particularly when the arrival rate of local requests

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

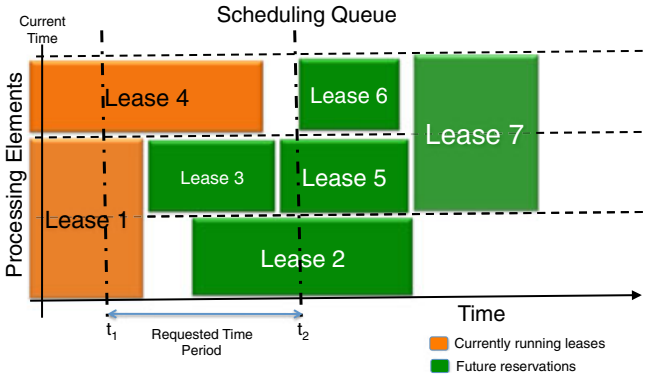


Figure 2. Resource contention takes place when the interval $t_1$ to $t_2$ is requested by a local requests. In this situation, both currently running leases and those waiting in the queue are affected.

is high. Additionally, precise estimation of the imposed overhead is necessary for implementing preemption-based resource scheduling [3].

Existing works on modeling the time overhead of VM preemption [1, 3] consider the amount of memory that should be deallocated, the number of VMs mapped to each physical node, and the use of local or shared storage. However, there are other factors such as communication between VMs that also have to be considered. Therefore, the second problem we address in this paper is to model the overhead time of preempting VM-based leases by taking into account the possible communication between VMs of leases.

The second side effect of preempting leases is increasing the makespan time of the external requests. Indeed, many of the current distributed systems use a variation of the backfilling policy for scheduling [12–14]. In the backfilling, future resource availabilities are reserved for requests that are waiting in the queue. Preempting leases and vacating resources for local requests can potentially affect these reservations. For instance, in Figure 2, vacating resources between $t_1$ and $t_2$ affect the currently running leases (leases one and four) as well as reservation waiting in the queue (leases two, three, and five). The affected leases and reservations are scheduled at a later time in the scheduling queue. Therefore, preempting external leases delays their execution and increases their makespan time.

By getting lease preemption possible in an RP, there is a possibility that several leases have to be preempted to free sufficient resources for an arriving local request. Therefore, there are potentially several sets of candidate leases that can be preempted. We term each set of the candidate leases a *Candidate Set*. For example, in Figure 2, if a local request needs one VM for the duration of $t_1$ to $t_2$, then the candidate sets are leases $\{1, 2\}, \{1, 3, 5\}, \{4\}$. Selecting different candidate sets for preemption affects the amount of imposed overhead as well as the makespan of external requests. Therefore, the third problem we address in this paper is how to choose an optimal candidate set for preemption in a way that the side effects of preempting leases are reduced.

In summary, this paper makes three contributions as follows:

(i) Facilitating preemption of VM-based leases to allocate local requests.
(ii) Modeling the imposed overhead of preempting VM-based leases with respect to the performed operation on the VMs.
(iii) Proposing preemption policies to determine the optimal set of leases for preemption.

### 2.1. *Problem formulation*

The formal definition of the problem can be stated as follows:

- $L_i$: Lease $i$.
- $R_j$: Local request $j$
- $\tau(L_i) \in \mathit{LeaseTypes}$ (see Section 4.2)

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

- $v(x_i)$: Number of VMs in the lease/request $i$.
- $h(L_i)$: Overhead time of preempting lease $i$.
- $p(L_i)$: Category of lease $i$ (local or external) and defined as follows:

$$p(L_i) = \begin{cases} 1 & \text{if external request} \\ 0 & \text{if local request} \end{cases}$$

According to the aforementioned definitions, candidate set $m$ for allocating local request $R_j$ can be presented as follows:

$$C_m : \left\{ L_i \mid p(L_i) = 1 \;\&\; v(R_j) \leqslant \sum_{i=1}^{N_m} v(L_i) \right\} \tag{1}$$

where $N_m$ is the number of leases involved in candidate set $C_m$. If there are $S$ candidate sets, then all candidate sets can be presented as follows:

$$A : \{C_m \mid 0 \leqslant m \leqslant S-1\} \tag{2}$$

Finally, a preemption policy can be presented as a function that selects an appropriate candidate set out of all candidate sets (i.e. $policy(A) = C_m$).

We believe that in an RP with many nodes and requests, selecting a proper candidate set for preemption is crucial and leads to significant reduction in preemption overhead time and increases user satisfaction. Although the problem we are investigating is in InterGrid context, it could be also applied to other lease-based Grid/Cloud RPs where requests with higher priority (such as local or organizational requests) coexist with other requests.

Specifically, the environment and assumptions we consider in this research share similarities with current public Cloud providers. One important similarity is that our proposed preemption policies do not have any assumption or estimation about the leases duration. However, there are differences that can be considered in a future research direction. Most importantly, prioritizing leases in public Cloud providers are economy driven (e.g Spot, On-demand, and Reserved instances in Amazon EC2 have different prices and different priorities). Another difference is that public Clouds are primarily tailored for transactional workloads [15], such as web transactions. Nonetheless, leases we consider in this research are in form of batch requests.

## 3. RELATED WORK

Although preemption was not extensively studied in distributed computing previously [12], emergence of resource provisioning based on VMs has motivated many research works to be undertaken in the area.

Haizea [8] is a lease scheduler that schedules advance-reservation and best-effort leases. Haizea preempts best-effort leases in favor of advance-reservation requests. In case of preemption, the basic preemption policy in Haizea tries to affect (preempt) the minimum possible number of leases (see Section 6.2 for detailed description). By contrast, we propose and compare policies that determine the optimal candidate set for preemption on the basis of its imposed overhead and the number of leases affected. In Haizea, all preempted leases are suspended and put in the queue to be resumed later, whereas we consider a diversity of lease types that enable more operations, such as migration of the leases.

In another research undertaken by Sotomayor *et al.* [3], the overhead time imposed for suspending and resuming a VM-based lease is estimated. The proposed model is based on the amount of memory that should be deallocated, the number of VMs mapped to each physical node, the local or global memory used for allocating VMs, and the delay related to commands being enacted. Nonetheless, they have not considered situation where there is communication between VMs of a lease.

Walters *et al.* [2] introduced a preemption-based scheduling policy for combination of batch and interactive jobs within a Cluster. In this work, batch jobs are preempted (suspended) in favor of

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

interactive jobs. The authors introduce different challenges in preempting jobs including selecting a proper job to be preempted, checkpointing of the preempted job, and VM provisioning. Their preemption policy is based on weighted summation of several factors such as the time spent in the queue. By contrast, we consider various operations for preemption and model the overhead of performing these operations.

Kettimuthu *et al.* [16] focused on the impact of preempting parallel jobs in supercomputers for improving their average and worst case slow down. The authors suggested a preemption policy, called selective suspension, where an idle job can preempt a running job if its suspension factor is adequately more than the running job.

However, the authors do not specify which job should be preempted; instead, they decide when to do the preemption. The proposed policy is starvation-free because it operates based on the response ratio of jobs.

Isard *et al.* [17] investigate the problem of optimal scheduling for data intensive applications, such as Map-Reduce [17], on the Clusters that computing and storage resources are close together. They propose a scheduling policy that preempts currently running job to maintain data locality for a new job. Although the scheduling policy is based on job preemption, the authors do not discuss which job is selected to be preempted amongst several candidates.

Snell *et al.* [12] consider the impact of preemption on the backfilling scheduling. They provide policies to select a set of jobs for preemption in a way that jobs with higher priority are satisfied and the resources utilization increases. In this work, the preempted job is restarted and rescheduled in the next available time. Our work is different than Snell *et al.* from several aspects. Firstly, we consider VM-based leases that offer more choices for the preempted lease. Secondly, Snell *et al.* recognize the best set of running jobs for preemption, whereas in our contribution, the preemption policy considers current leases as well as the reservations waiting in the queue. The third difference is that they do not consider the overhead of preempting jobs. In fact, by killing the preempted jobs, they reduced the overhead to zero. Nonetheless, the computational power is wasted in that case.

## 4. ARCHITECTURAL FRAMEWORK

In this section, we briefly overview the InterGrid architecture and the required extension in terms of different lease types for the proposed solutions. This section also provides necessary background for the implementation explained in Section 8.

### *4.1. InterGrid architecture*

InterGrid provides an architecture and policies for federating Grids [6]. Figure 3, illustrates a scenario in which multiple Grids, are federated through IGGs. A Grid has predefined peering arrangements [6] with other Grids, through which IGGs coordinate adoption of resources from other Grids. An IGG is aware of the peering terms between Grids, selects suitable Grids that can provide the required resources, and replies to requests from other IGGs. In InterGrid, each request is contiguous and has to be served within resources of a single RP.

The local resource manager system (LRMS)[‡] is the resource manager in each RP that provisions resources for local and external requests.

The main part of InterGrid architecture is the IGG. The core component of IGG is its *Scheduler*, which implements provisioning policies and peering with other IGGs. The scheduler orders creation, starting, and stopping of VMs through the VM manager (VMM). VMM implementation is generic; so, different LRMSs (in different RPs) can interact with it. Currently, it is possible for VMM to connect to OpenNebula [18], or Eucalyptus [19] to manage the local resources. In addition, two interfaces to connect to a Grid middle-ware (i.e. Grid'5000 [4]) have been developed. Moreover, an emulated LRMS for testing and debugging has been implemented for the VMM.

[‡]This component is also called virtual infrastructure engine in the InterGrid.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

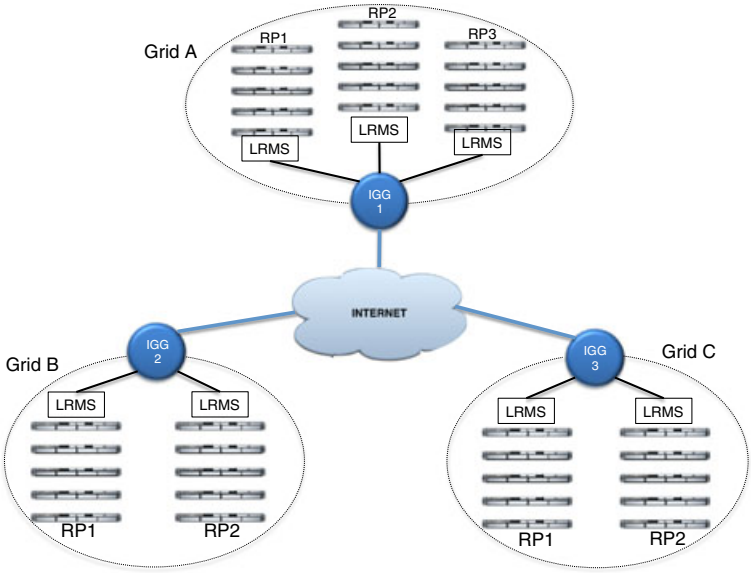


Figure 3. A high-level view of InterGrid architecture. LRMS, local resource manager system; RP, resource provider; IGG, InterGrid Gateway.

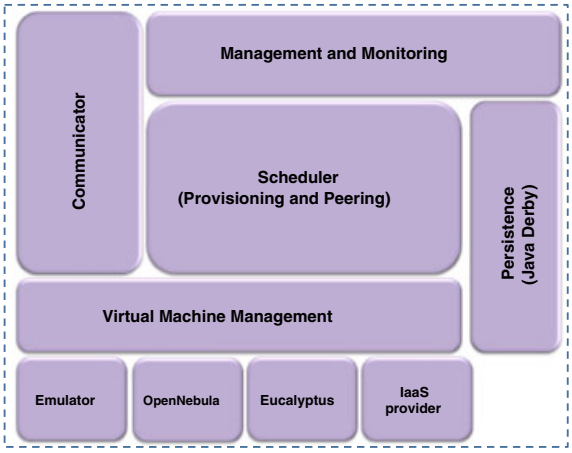


Figure 4. Internal structure of the InterGrid Gateway. (IaaS: Infrastructure as a Service).

The internal architecture of IGG is illustrated in Figure 4. The persistence database is used for storing information of IGG such as VM templates and peering arrangements. The management and monitoring provide command-line tools to configure and manage IGG. The communication module provides an asynchronous message-passing mechanism between IGGs, which makes IGGs loosely coupled and fault-tolerant.

### *4.2. Introducing different lease types*

To tackle the challenges mentioned in Section 2, we should first make the preemption possible in lease terms agreed between RP and consumer in InterGrid. In fact, one difference between job-based and lease-based resource provisioning is that jobs can be preempted without notifying the job owner. However, this is not the case for leases [7]. Therefore, to make preemption possible for leases, in this section, we come up with regulations in the lease terms. For that purpose, we introduce different request types in InterGrid [11].

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Currently, a request issued by a user in InterGrid has the following characteristics:

- VM template needed by the user.
- Number of VMs.
- Ready time is the time that requested VMs should be ready.
- Deadline is the time that serving the request must be finished (this is an optional field).

We extend the InterGrid request by adding the 'request type' to it; therefore, users can select type of their requests. When a reservation is created for a user request, the 'request type' is mapped to 'lease type'.

On the basis of the request/lease type, the scheduler determines how to schedule the lease and which operation should be performed on the preempted lease(s).

Different request types we consider for requests in InterGrid are broadly classified as best-effort and deadline-constraint requests. More details of different request types are as follows:

- *Best-Effort-Cancellable*: These requests can be scheduled at any time after their ready time. Leases of such type can be canceled without notifying the lease owner. Cancellable leases neither guarantee the deadline nor the duration of the lease. Such leases are suitable for map-reduce-like requests [17]. Spot instances [20] in Amazon EC2[§] is another example of Cancellable leases.
- *Best-Effort-Suspendable*: Leases of this type can be suspended at any time but should be resumed later. This type of lease guarantees the duration of the lease but not in a specific deadline. Suspendable leases are flexible in start time and can be scheduled at any time after their ready time. In the case of preemption, these leases should be rescheduled to find another free time-slot for the remainder of their execution. Suspendable leases are suitable for Bag-of-task [21] and Parameter Sweep type of applications [22].
- *Deadline-Constraint-Migratable*: These leases guarantee both the duration and deadline of the lease. However, there is no guarantee that they will be run on a specific resource(s). In other words, there is a chance for the lease to be preempted and migrated but it will be resumed and finished before its deadline, either on the same resource or on another resource. Migratable leases are needed by applications with tight Quality of Service (QoS) demands [23]. For these applications, the lease can be migrated to a more powerful RP to meet QoS constraints of the user, such as deadline.
- *Deadline-Constraint-Non-Preemptable*: The leases associated with such requests cannot be preempted at all. These leases guarantee both deadline and duration without being preempted during the execution of the lease. This type of lease is useful for critical tasks in workflows where some tasks have to start and finish at exact times to prevent delaying the execution of the workflow [24].

We assume that local requests are all deadline-constraint and non-preemptable. However, external users can send all request types mentioned earlier. In practice, different request types correspond to different prices. Thus, users with various demands are motivated to associate their requests to different request types. Unarguably, the more flexible request type, the less expensive the lease [25]. However, we leave the market-oriented implications of the lease-based scheduling as a future work.

## 5. MEASURING THE OVERHEAD OF PREEMPTING A VM-BASED LEASE

The time overhead imposed for preempting a candidate set depends on the type of leases involved within that candidate set. In other words, the overhead is driven by the operation performed on the VMs of leases involved in a candidate set. In this part, we provide the *worst-case* model for time overhead of possible operations on the VMs of a lease.

Preempting *cancellable* leases imposes the minimum time overhead. This overhead pertains to the time needed to stop VMs of a lease. The duration of the stop operation is independent from the

[§]http://aws.amazon.com/ec2/spot-instances

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

VM characteristics, such as memory size of the VM, and it is almost instant [1]. In practice, this time is usually negligible [3].

Preempting a *suspendable* lease is more complicated than a cancellable lease. One complexity relates to the message exchange between the VMs of the lease. In fact, running a distributed application inside the VMs of a lease implies exchanging messages between the VMs. However, suspending a lease occurs in the VM level that is unaware of the communication. Therefore, suspending a lease can interrupt the message communication and lead to inconsistent state for jobs running inside VMs [1].

More specifically, messaging is commonly performed through the Transmission Control Protocol to assure the message delivery. If the sender host does not receive acknowledgment after a certain number of retransmissions, then the connection with the receiver is terminated. At the suspension and resumption times, there is a possibility that some VMs become unreachable and consequently some connections are lost. Therefore, it is important to coordinate the suspend and resume operations to avoid inconsistent situation.

One possible way to reduce the impact of the suspend and resume operations on the jobs running within the VMs is *pausing* them before suspending and *unpausing* VMs after resumption. Pausing a VM prevents it from accessing the processor and is supported by hypervisors such as Xen [26]. This operation is quicker (takes few milliseconds) than writing the whole memory into the disk and can be completed before the unreachable delay of Transmission Control Protocol [1].

Taking these coordinations into consideration, the time overhead of preempting a suspendable lease broadly includes the time to pause VMs, suspending (i.e. writing the memory image of the VMs to the disk), and rescheduling the lease. Accordingly, resuming the lease includes the time for VM resumption (i.e. the time for loading the VMs' memory image from the disk) and then unpausing VMs.

Because pausing and unpausing operations take a short time, usually to control the order of these operations, they are performed sequentially on VMs (e.g. based on the VM identifier) [1]. If a lease $L_i$ contains $v(L_i)$ VMs, then the time for pausing the VMs is $v(L_i)\cdot t_p$ where $t_p$ is the time to pause a single VM. In our analysis, we consider the same amount of time for pausing and unpausing operations. Therefore, the overall time overhead of pausing and unpausing is $v(L_i)\cdot 2t_p$.

Suspending a lease implies rescheduling it for the remainder of the execution. The time overhead of rescheduling depends on the time complexity of rescheduling algorithm. The complexity usually depends on factors such as number of physical nodes and current workload condition. Nonetheless, because our model does not assume any particular scheduling policy, we consider a constant value ($\delta$) for the rescheduling time overhead.

The major time overhead in suspendable leases pertains to the time for suspending and resuming VMs. The time for these operations is driven by the memory size of each VM. Specifically, the suspension time for $\mathrm{VM}_j$ (which is shown as $t_s^j$) and resumption time ($t_r^j$) are worked out on the basis of Equations (3) and (4), respectively.

$$t_s^j = \frac{\mathrm{mem}_j}{B} \tag{3}$$

$$t_r^j = \frac{\mathrm{mem}_j}{r} \tag{4}$$

where $\mathrm{mem}_j$ is the memory size of $\mathrm{VM}_j$, $B$ is the rate of suspending megabytes of VM memory per second, and $r$ is the rate of re-allocating megabytes of VM memory per second [3].

We consider a shared storage in the RPs to be able to resume the suspended lease on any of its hosts. In this circumstance, to avoid any possible contention on the shared storage, suspension and resumption is when these operations are performed sequentially for all VMs of the lease. Therefore, for lease $L_i$ with $v(L_i)$ VMs, the suspension time overhead is $\sum_{j=1}^{v(L_i)} t_s^j$.

It is worth noting that this is the worst-case analysis for the suspension and resumption time overheads. We expect that the best and average analyses that considers factors, such as an overlapping, result in lower overheads for these operations.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

By taking into account all the aforementioned factors, the overall time overhead for suspending ($s(L_i)$) and resuming leases ($r(L_i)$) are calculated according to Equations (5) and (6), respectively.

$$s(L_i) = v(L_i)\cdot t_p + \sum_{j=1}^{v(L_i)} t_s^j + \delta \tag{5}$$

$$r(L_i) = v(L_i)\cdot t_p + \sum_{j=1}^{v(L_i)} t_r^j \tag{6}$$

where $t_p$ is the time for pausing a VM, $t_s^j$ is the time overhead of suspending $j$th VM of a lease, and $t_r^j$ is the time overhead of resuming $j$th VM of a lease. Therefore, the overall time overhead of preempting a suspendable lease ($h(L_i)$) is as follows:

$$h(L_i) = 2v(L_i)\cdot t_p + \delta + \sum_{j=1}^{v(L_i)} \left(t_s^i + t_r^j\right) \tag{7}$$

Time overhead imposed for preempting *migratable* leases includes VM image transferring overhead in addition to all overheads considered for suspendable leases [10]. More specifically, migrating a VM includes suspending it on source host, transferring the suspended VM to destination RP, and resuming VM in destination host. The overheads pertain to pausing, unpausing, and rescheduling VMs (i.e. finding a proper destination) also have to be taken into account.

In the transferring phase, disk memory image of suspended VM is transferred to destination host over the network [10]. The time for transferring depends on the size of the suspended VM and the network bandwidth, therefore, $t_{\text{copy}}^j = \text{mem}_j/b$ where $\text{mem}_j$ is the size of disk memory image for $VM_j$ and $b$ is the network bandwidth.

In migrating VMs of a lease, suspending VMs in the source RP can be overlapped with resuming them in the destination RP. Particularly, although the second VM is being suspended in the source RP, the first VM that has already been transferred to the destination RP can be resumed without conflicting with other operations. The overhead of these operations for $j$th VM of the lease is driven by the max $\left\{t_s^j, t_r^{j-1}\right\}$. Additionally, the time for suspending the first VM ($t_s^1$) and resuming the last VM of the lease $\left(t_r^{v(L_i)}\right)$ cannot be overlapped. Thus, the overall time for suspend and resume phases of migrating VMs for lease $i$ is $t_s^1 + \sum_{j=1}^{v(L_i)-1} \max\left\{t_s^j, t_r^{j-1}\right\} + t_r^{v(L_i)}$.

Additionally, the time for transferring VMs ($t_{\text{copy}}$) cannot be overlapped and has to be carried out sequentially for all the VMs to avoid any contention on the shared storage. Therefore, the overall time for transferring phase of migrating a lease is as follows: $\sum_{j=1}^{v(L_i)} t_{\text{copy}}^j$.

The overheads regarding rescheduling, pausing, and unpausing VMs are the same as those discussed for suspendable leases. The overall overhead of migrating VMs of lease $L_i$ can be worked out on the basis of Equation (8).

$$h(L_i) = \sum_{j=1}^{v(L_i)} t_{\text{copy}}^j + t_s^1 + \sum_{j=1}^{v(L_i)-1} \max\left\{t_s^j, t_r^{j-1}\right\} + t_r^{v(L_i)} + 2v(L_i)\cdot t_p + \delta \tag{8}$$

It is worth noting that we assume that the destination host has the disk image of the VM; therefore, it is not needed to be transferring over the network.

## 6. PREEMPTION POLICIES

When an LRMS cannot find sufficient vacant resource for a local request, it forms all candidate sets where each candidate set contains leases that their preemption creates enough space for an arriving local request. Preemption policy in this situation determines the proper candidate set for preemption.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

From the user perspective, selecting different candidate sets is decisive for the amount of resource contention that take place within an RP between local and external requests. Additionally, it affects the waiting time of external requests. From the system centric perspective, choosing different candidate sets determines the number of VMs to be preempted as well as the operation that has to be performed on them (e.g., suspension or migration). Therefore, choosing different candidate sets influences the amount of time overhead imposed to the system.

In the following, we evaluate the impact of various preemption policies in terms of time overhead and resource contention. The first policy focuses on the system centric criteria by trying to minimize time overhead and, consequently, increase resource utilization. The second policy focuses on user centric criteria and sought to minimize the resource contention by preempting fewer leases and affecting fewer users. The third policy makes a trade-off between resource utilization and user satisfaction.

### *6.1. Minimum overhead policy*

As a system centric policy, this policy aims at maximizing resource utilization. Therefore, this policy tries to minimize the time overhead imposed to the underlying system by preempting a candidate set that leads to the minimum overhead. For this purpose the total overhead imposed to the system by each candidate set is calculated and a set with minimum overhead is selected. Minimum overhead (MOV) policy can be formally presented on the basis of Equation (9).

$$MOV(A) = \min_{m=0}^{S-1}\{h(C_m)\} \tag{9}$$

### *6.2. Minimum leases involved policy*

Preempting leases makes longer waiting times for suspendable and migratable leases to get completed. In the case of cancellable leases, preempting the lease results in terminating the lease. Therefore, as a user centric policy, minimum leases involved policy (MLIP) tries to satisfy more users by creating less resource contention and preempting fewer leases.

In this policy, a candidate set that contains the minimum number of leases is selected from all the candidate sets regardless of lease types involved in each candidate set. MLIP can be presented according to Equation (10).

$$MLIP(A) = \min_{m=0}^{S-1}\{|C_m|\} \tag{10}$$

where $|C_m|$ gives the number of leases involved (cardinality) in each candidate set $C_m$. This policy is practically used as the preemption policy in the Haizea [8] scheduler, and we use it as the *baseline* policy to evaluate the performance other preemption policies (Section 7).

### *6.3. Minimum overhead minimum lease policy*

The two policies mentioned earlier aim to either improve resource utilization (as a system centric criterion) or minimize the number of lease preemption (as a user centric criteria). However, the minimum overhead minimum lease (MOML) policy fulfills both system and user centric criteria at the same time. The way this policy operates is depicted in Figure 5, and its pseudo code is shown in Algorithm 1. In fact, MOML makes a trade-off between MOV, which minimizes the imposed overhead, and MLIP, which minimizes the number of resource contentions.

According to Figure 5 and Algorithm 1, in MOML, the selection of a candidate set is carried out in two phases. In the first phase (pre-selection), all candidate sets that have a total overhead less than a certain threshold ($\alpha$) are pre-selected for the second phase (lines five to eight in Algorithm 1). The pre-selection phase increases the tolerance of acceptable overhead in comparison with MOV. In the second phase, to have fewer resource contentions, a candidate set that contains minimum number of leases is selected (lines 9 to 11 in Algorithm 1).

Selecting a proper value for $\alpha$ determines the behavior of MOML policy. More specifically, if the $\alpha \to \infty$, then MOML behaves the same as MLIP. On the other hand, if $\alpha \to 0$, then MOML

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

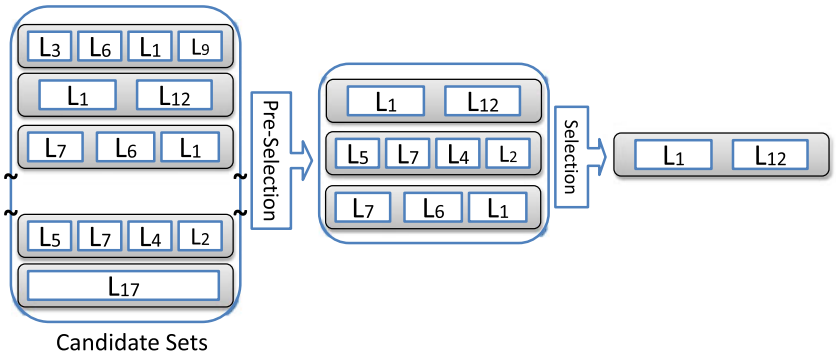


Figure 5. Pre-selection and final selection phases of minimum overhead minimum lease policy.

**Algorithm 1:** Minimum overhead minimum lease preemption policy (MOML).

**Input**: Candidate Sets
**Output**: Selected Candidate Set
1 **foreach** $candidateSet \in$ *Candidate Sets* **do**
2  `Overheads.Add(getOverhead(`$candidateSet$`));`
3 min $\leftarrow \infty$;
4 $\alpha \leftarrow$ `getMedian(`*Overheads*`);`
5 **foreach** $candidateSet \in$ *Candidate Sets* **do**
6  $ovhd \leftarrow$ `getOverhead(`$candidateSet$`);`
7  $NoLeases \leftarrow$ `Cardinality(`$candidateSet$`);`
8  **if** $ovhd \leqslant \alpha$ **then**
9  **if** $NoLeases <$ min **then**
10  $selected \leftarrow candidateSet$;
11  min $\leftarrow NoLeases$;

behaves the same as MOV. Thus, to keep the trade-off between MOV and MLIP, we consider $\alpha$ as the *median* value of the overheads (lines one, two, and four in Algorithm 1). By choosing $\alpha = median$ we ensure that just half of the candidate sets that have lower overheads are considered in the second phase for having a minimum number of leases.

## 7. PERFORMANCE EVALUATION

In this section, we discuss different performance metrics considered, the scenario in which the experiments were carried out, and experimental results obtained from simulation.

### *7.1. Performance metrics*

Introducing different types of leases along with preemption policy are expected to affect different parameters, which are described over the next subsections.

*7.1.1. Local and external request rejection rate.* The initial objective of this paper is to serve more local requests by preempting resources from external leases. Therefore, it is interesting to realize the efficiency of different preemption policies in terms of serving more local requests.

We define the 'local request rejection rate' that is the fraction of local requests that are rejected, possibly because of allocating resources to non-preemptable external requests or other local requests.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Additionally, we are interested to see if decreasing local request rejection rate comes with the cost of rejecting more external requests. External request rejection rate describes this metric and shows the percentage of external requests that are rejected. The ideal case is that local request rejection rate is reduced without increasing the external request rejection rate.

*7.1.2. Resource utilization.* Time overhead is a side-effect of preempting VMs that degrades resource utilization. Therefore, we are interested to see how different preemption policies affect the resource utilization. Resource utilization is defined according to Equation (11).

$$Utilization = \frac{computationTime}{totalTime} \times 100 \tag{11}$$

where:

$$computationTime = \sum_{i=1}^{|\lambda|} v(L_i) \cdot d(L_i) \tag{12}$$

where $|\lambda|$ is the number of leases, $v(L_i)$ is the number of VMs in lease $L_i$, and $d(L_i)$ is the duration of the lease $L_i$.

*7.1.3. Number of lease preemption.* Total number of lease preemptions is a proper metric to measure user satisfaction resulting from different preemption policies.

*7.1.4. Makespan time.* Makespan is a user centric metric that is affected by preemption. This metric is prominent for best-effort external requests that are in the risk of getting preempted several times that increases their makespan. This metric measures the amount of time; on-average, a best-effort lease should wait beyond its ready time to be completed. Makespan is calculated on the basis of Equation (13).

$$Makespan = \frac{\sum_{L_i \in \beta} (c(L_i) - r(L_i))}{|\beta|} \tag{13}$$

where $\beta$ is the set of best-effort leases, $|\beta|$ is the number of best-effort leases, and $c(L_i)$ and $r(L_i)$ show the completion time and ready time of lease $L_i$, respectively. Although best-effort requests are not bound to any deadline, users are more satisfied to wait less for their requests to be completed.

### *7.2. Experimental setup*

For simulation, we used Gridsim [27] as a discrete event simulator. In the experiments conducted, Lublin99 [28] has been configured to generate a two-week-long workload that includes 3000 parallel requests. Lublin99 is a workload model on the basis of the San Diego Super Computer Blue Horizon machine. Job traces collected from this supercomputer are publicly available and have been studied extensively in the past.

To simulate an RP within InterGrid, we consider a Cluster with 32 worker nodes. We assume all nodes of the RP as single core with one VM. We also assure that the number of VM(s) needed by requests would not be more than Cluster nodes. It is worth noting that our proposed model and policies are not limited to this configuration and can support multicore architectures and several VMs on each worker node.

We consider each VM of 1024 MB and a 100 Mbps network bandwidth. We also assume a shared file system (e.g., Network File System (NFS)) for the Cluster where the disk images for VMs and memory snapshots for suspended VMs are maintained. We assume each VM disk image is 2 GB. Because we consider that the disk images are replicated on all RPs in the InterGrid, they are not needed to be transferred.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

On the basis of the research by Sotomayor *et al.* [3] and considering the 100 Mbps network bandwidth, the suspending rate of VM memory as $s = 6.36$ MB/s, and re-allocating rate as $r = 8.12$ MB/s (Section 5). Hence, in our experiments, suspension time ($t_s$) and resumption time ($t_r$) for a lease with 1 VM are 161.0 and 126.1 s, respectively. These values were also validated through experimenting in the real environment. In migrating a VM with similar configuration, the time overhead for transferring a suspended VM to another RP ($t_{copy}$ in Equation (8)) of InterGrid is 160.2 s [10]. On the basis of our experiments, pausing and unpausing operations on each VM takes 5 ms. Finally, we employ a conservative backfilling as the scheduling policy in the LRMS. During the initial experiments in the real environment, we noticed that the average overhead time of rescheduling is 2.3 s.

We study the behavior of different policies when they face workloads with different characteristics. For that purpose, we modify the characteristics of the workloads when:

- Percentage of best-effort external requests (i.e., cancellable and suspendable) varies.
- Percentage of deadline-constraint external requests (i.e., migratable and non-preemptable) varies.
- Percentage of local requests varies.

Because Lublin workload does not provide us the request types, we generated these types uniformly and assigned them to the generated workloads. We changed the percentage of best-effort and deadline-constraint requests from 10% to 50% of the external requests, whereas the number of local requests remains constant (1000). In other configuration, local requests are also changed from 20% to 70% of the whole workload. In fact, we experimented conditions that local requests are below and above these limits. However, we noticed that not many preemption take place in those points; therefore, there is not major difference between policies. To have a realistic evaluation, in the Lublin workload, we adjusted the average number of VMs of leases to be 4 and the average duration of requests to be 2 hours.

### *7.3. Results and discussions*

*7.3.1. Local and external request rejection rate.* In this experiment, we evaluate the efficacy of the preemption mechanism in resolving the resource contention between local and external requests. For that purpose, we compare performance of the MOML policy against the baseline situation, that is, when there is no preemption mechanism in place. More specifically, we report the difference between local requests' rejection rate in these two situations.

In Table I, the mean difference of decrease in local requests rejection rate is reported (in the second column) along with a 95% confidence interval of the difference (in the third column). We use a *t*-test to work out the mean difference between these two policies. To perform the *t*-test, we have ensured that the distribution of difference is normal. Each row of the table shows results of the experiment when one characteristic of the workload, which is shown in the first column, is changed.

Table I. Mean difference and 95% confidence interval (CI) of decrease in local requests rejection rate and external requests rejection rate as a result of applying preempting leases in a resource provider of InterGrid.

| **Modified parameter** | **Mean decrease in Requests rejection rate** | **CI of decrease in local requests Rejection rate** | **Change in external Rejection requests rate** |
|---|---|---|---|
| Percentage of BE external requests | **72.0**% | (51.1, 92.8), $p$-value=0.001 | Not statistically significant, $p$-value=0.6 |
| Percentage of DC external requests | 54.3% | (35.0, 73.7), $p$-value=0.001 | Not statistically significant, $p$-value=0.3 |
| Percentage of local requests | 58.2% | (40.3, 75.9), $p$-value<0.001 | Not statistically significant, $p$-value=0.6 |

BE, best-effort; DC, Deadline-Constraint.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

According to the second column in Table I, local request rejection rate has statistically and practically significantly decreased by applying preemption mechanism in comparison with situation that there is no preemption mechanism in place. The 95% confidence interval and the corresponding $p$-value of the difference in rejection ratio of local requests is reported in the third column of Table I. More importantly, this reduction in local request rejection rate has not been with the cost of rejecting more external requests. On the basis of the fourth column in Table I, in all experiments, external request rejection rate does not change significantly.

On the basis of this experiment, the maximum decrease in the local request rejection rate occurs when the percentage of best-effort external requests is higher (the first row in Table I). In this circumstance, more local requests can be accommodated by preempting the best-effort leases.

*7.3.2. Resource utilization.* In this experiment, we measure the resources utilization when different preemption policies are applied. In all subfigures of Figure 6, it is observed that MOV results in better utilization comparing with the other policies. However, in a few points (e.g. in Figure 6(a) when 40% of the requests are best-effort), MOV has slightly less utilization than MOML. The reason is resource fragmentations (i.e. unused spaces) in the scheduling queue that leads to resource underutilization. Subfigures of Figure 6 also demonstrates that resource utilization MOML lies between MLIP and MOV.

Figure 6(a) indicates that increasing the percentage of best-effort requests improves the resource utilization; however, after a certain point (i.e. best-effort>20%), resource utilization does not fluctuate significantly in different policies. Indeed, in this situation, unused spaces are allocated to the preempted leases.

In Figure 6(b), we can see that resource utilization increases by increasing the percentage of deadline-constraint requests in all policies. In fact, having more deadline-constraint requests imply

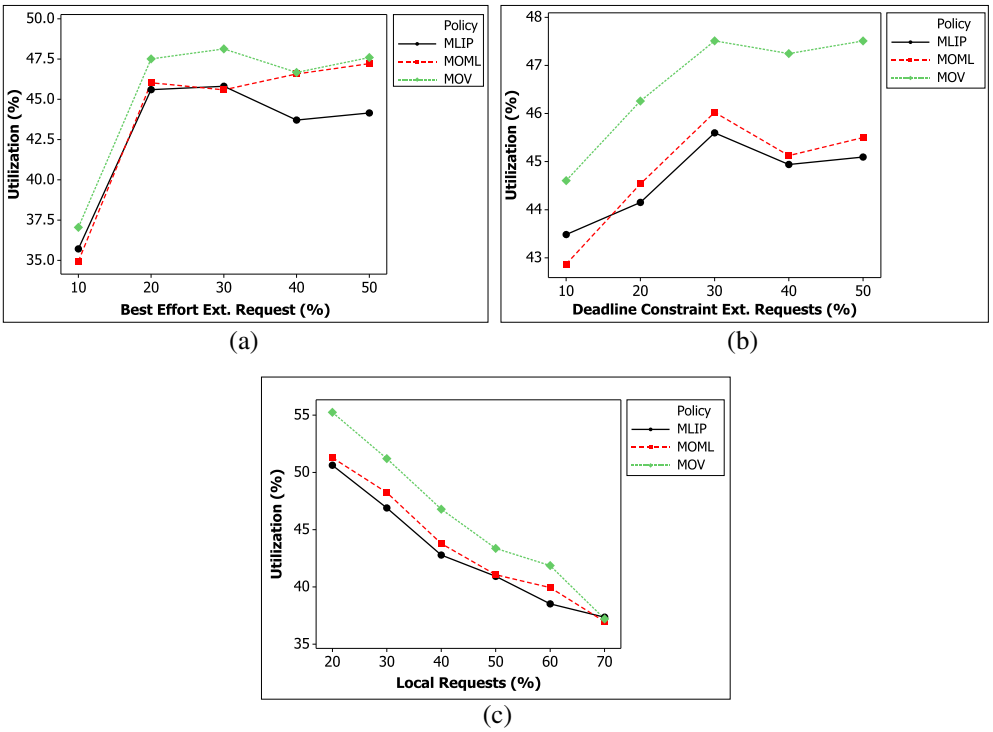


Figure 6. Resource utilization results from different policies. The experiment was carried out by modifying (a) the percentage of best-effort external requests, (b) the percentage of deadline-constraint external requests, and (c) the number of local requests. MLIP,minimum leases involved policy; MOML,minimum overhead minimum lease; MOV, minimum overhead.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

fewer preemptions and more resource utilization. As expected, MOV policy outperforms other policies because of preempting leases that impose less overhead.

In Figure 6(c), it is expressed that by increasing the percentage of local requests, the number of preemption and subsequently the amount of overhead is increased. Therefore, resource utilization decreases almost linearly in all policies. Another reason for the reduction in resource utilization is that local requests are not preemptable and their scheduling leads to many fragmentation in the scheduling queue.

*7.3.3. Number of lease preemptions.* The number of external leases that are preempted in different preemption policies indicates the amount of (external) user dissatisfaction in the system.

Figure 7(a) shows that when the percentage of best-effort requests is increased, the number of preemptions rises almost linearly. For the lower percentages of best-effort external requests (best-effort<30%), MOML behaves similar to MOV; however, after that point, MOML approaches MLIP. The reason is that when the percentage of best-effort leases is high, the likelihood of having a candidate set with the minimum number of leases and not large overall overhead is high. Thus, MOML approaches MLIP.

Figure 7(b) demonstrates that the number of preemptions does not vary significantly when the percentage of deadline-constraint requests is less than 40%. In fact, in this situation, there are enough best-effort requests for preemption, and changing the percentage of deadline-constraint requests does not play an important role.

Figure 7(c) reveals the impact of number of local requests on the resource contention. We can see that in all policies, the number of lease preemptions is increased almost linearly by increasing the percentage of local requests.

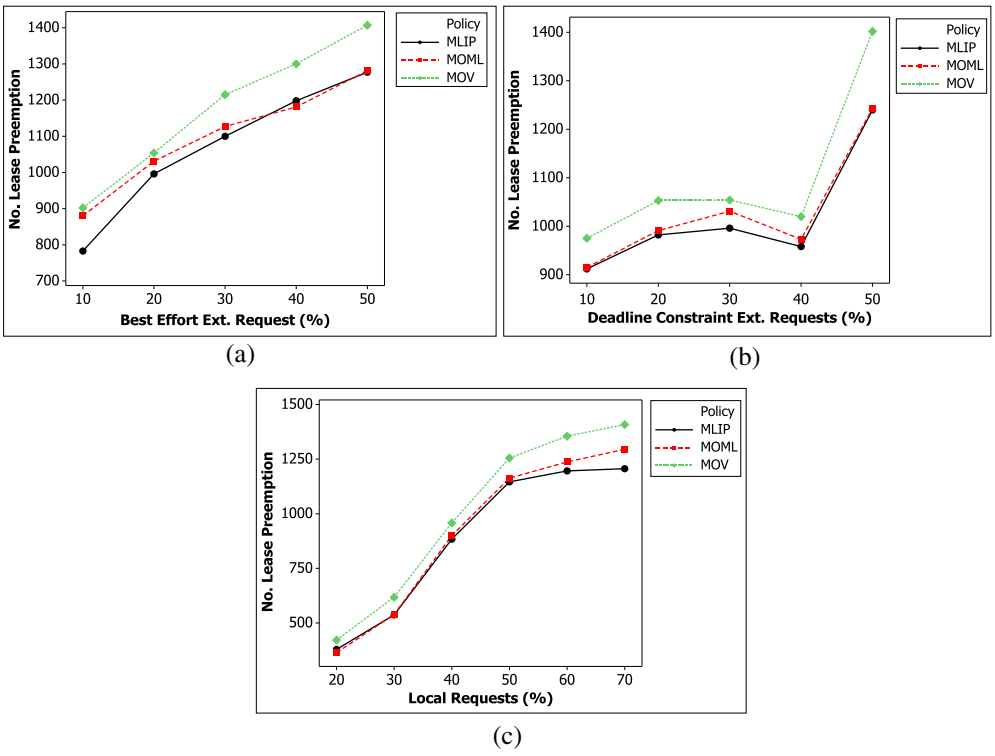


Figure 7. Number of lease preemption resulted from different policies by changing (a) percentage of best-effort external requests, (b) percentage of deadline-constraint external requests, and (c) the number of local requests. MLIP, minimum leases involved policy; MOML, minimum overhead minimum lease; MOV, minimum overhead.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

In general, in all subfigures of Figure 7, MLIP results in fewer number of lease preemptions (resource contention), and MOML operates between MLIP and MOV.

*7.3.4. Average makespan time.* In this experiment, we investigate the impact of different preemption policies on the average makespan of the best-effort external requests. The results of the experiment under different workloads are illustrated in Figure 8. All subfigures of Figure 8 testify that MLIP leads to less makespan time in comparison with other policies. The reason is that MLIP disregards the type of leases for preemption. This means that, comparing with MOV, it is less likely in MLIP to preempt best-effort requests. Therefore, the best-effort requests are completed earlier, and their average makespan time is lower in MLIP. Figure 8(a) demonstrates that by increasing the percentage of best-effort requests, the average makespan time decreases after a certain point. When 20% of external requests are best-effort, because of numerous preemptions takes place, the average makespan time is in its peak. However, after that point, we notice a decrease in makespan time of the best-effort requests. This decrease occurs due to fewer deadline-constraint requests and more opportunities for local requests to be allocated. When 10% of the external requests are best-effort, because there are not many preemptable requests in the system, many local requests are rejected, and few preemption occurs. Hence, the average makespan time is low in that point.

Figure 8(b) shows that by increasing the percentage of deadline-constraint requests, the average makespan time decreases. In fact, increasing the percentage of deadline-constraint requests implies fewer best-effort external requests exist in the system. Therefore, the average makespan time for best-effort external requests is decreased.

Figure 8(c) illustrates that by increasing the percentage of local requests in the system (and consequently increasing the number of preemptions), the average makespan time is increased. As the percentage of best-effort requests is decreased by rising the percentage of local requests, we notice that makespan becomes flat when more than 50% of requests are local.

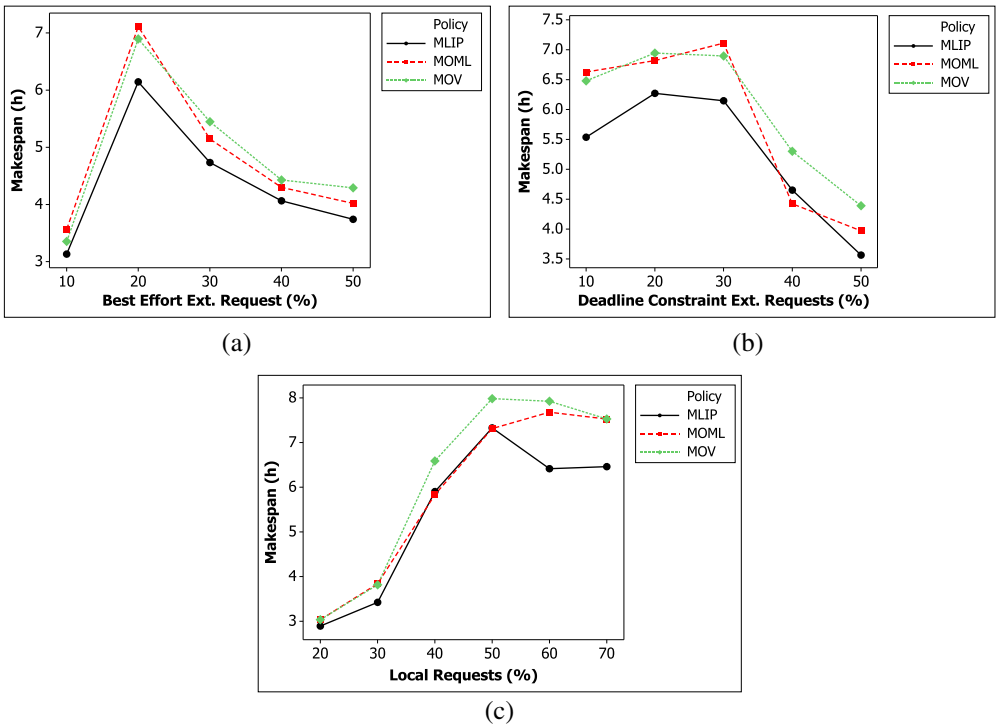


Figure 8. Average makespan time resulted from different policies. The experiment is carried out by altering (a) percentage of best-effort external requests, (b) percentage of deadline-constraint external requests, and (c) the number of local requests. MLIP, minimum lease involved policy; MOML, minimum overhead minimum lease; MOV, minimum overhead.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

## 8. IMPLEMENTATION OF THE PREEMPTION POLICIES IN THE INTERGRID

This section presents the realization of the investigated policies in the context of InterGrid. It specifically realizes the architecture described in Section 4.1.

### *8.1. Virtualization infrastructure*

In our implementation, we consider an RP as a compute Cluster. We employed OpenNebula [18] as the virtual infrastructure manager to handle the VMs lifecycle across the Cluster. OpenNebula provides a software layer on top of hypervisors and enables dynamic provisioning of resources to requests. The architecture of OpenNebula has been designed to be flexible and modular, and supports various hypervisors and infrastructure configurations within the Cluster. For each user request, OpenNebula starts, manages, and stops VMs according to the provisioning policies in place.

Along with the virtual infrastructure manager, a hypervisor is required in each node of the RP to provide VMs. In our implementation, we use Kernel-based VM (KVM) [29] as the hypervisor within each node of the Cluster. KVM is a hardware-assisted, fully virtualized platform for Linux on X86 hardware. By installing KVM, multiple execution environments (guest VMs from different disk images) can be created on top of each physical node. Each of these VMs has private, virtualized hardware, including a network card, storage, memory, and graphics adapter.

As mentioned earlier, OpenNebula as the virtual infrastructure manager in the Cluster offers an immediate provisioning model, where virtualized resources are allocated when they are requested. However, resource provisioning in InterGrid has requirements that cannot be supported within the immediate model. For instance, requests that are subject to priorities, capacity reservations at specific times, and variable resource usage throughout a VM's lifetime.

### *8.2. Local scheduler*

Haizea is an open source scheduler developed by Sotomayor *et al.* [8, 30] that employs VM-based leases for resource provisioning. The advantage of Haizea is considering priority between leases as well as the overheads of deploying VMs (e.g. suspending and resuming) in the scheduling. Therefore, we adopt Haizea as the local scheduler of the LRMS in RPs. As a result, the scheduling capability of the virtual infrastructure manager (i.e. OpenNebula) is extended and enables the LRMS to recognize the contention between local and external requests occurs in the RP.

Adopting Haizea as the local scheduler enables leasing resources to external requests and preempt them in favor of local requests to serve them within their requested time interval. In this way, the local scheduler operates as the scheduling back-end of OpenNebula. It also employs conservative backfilling algorithms along with VMs' abilities to efficiently schedule the leases and increase the resource utilization.

Although incorporating the preemption-based local scheduler into the architecture of InterGrid enables recognition of the contention between local and external requests, the scheduler does not consider the side effects implied by preemption. Therefore, in the next step, we implement different preemption policies as discussed in Section 6, within the local scheduler to detect the resource contentions and try to minimize their impact.

### *8.3. Evaluation scenario*

The testbed for evaluation of the implemented system is as follows:

- A four-node Cluster as the RP. Servers are three IBM System X3200 M3 machines, each with a quad-core Intel Xeon x3400, 2.7 GHz processor, and a 4 GB memory. The head node, where the LRMS resides, is a Dell Optiplex 755 machine with Intel Core 2 Duo E4500, 2.2 GHz processor, and 2 GB memory.
- The host-operating system installed in the server nodes is CentOS 6.2 Linux distribution. Also, the operating system in the head node is Ubuntu 12.4.
- All the nodes are connected through a 100 Mbps switched Ethernet network.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

- We used OpenNebula 3.4 and Haizea version 1.1 as the virtual infrastructure manager and the local scheduler, respectively.
- Qemu-KVM 0.12.1.2 is used as the hypervisor on each server.
- GlusterFS is used as the Cluster file system. It aggregates commodity storages across a Cluster and forms a large parallel network file system [31]. The disk images needed by the VMs and the memory image files (created when a VM is suspended) are stored on the shared file system.

The scenario we consider for evaluation involves an InterGrid with three IGGs with peering arrangements established between them, as illustrated in Figure 9. IGG1 has the Cluster as the RP and users from IGG2 and IGG3 request leases through the IGG interface. IGG1 receives these requests in the form of external requests and are allocated resources through the local scheduler in the LRMS of the RP. However, the RP has its own local requests that have more priority than the external ones. Information of the lease requests received by the LRMS are explained in Table II.

To be able to follow the order of events occurring in the system and demonstrate their impact, we perform the evaluation on few lease requests. According to the table, seven lease requests are submitted to the RP. Each row of the table shows the arrival time, the number of requested processing elements, the amount of memory, the duration, and the request type (i.e. local or external). We assume all external requests as BE-Suspendable. We consider 00:00:00 as the start of the experiment (i.e. the arrival of the first request), and the arrival time of other requests are proportional to the start time of the experiment. All of these lease requests use a ttylinux disk image located on the shared storage.

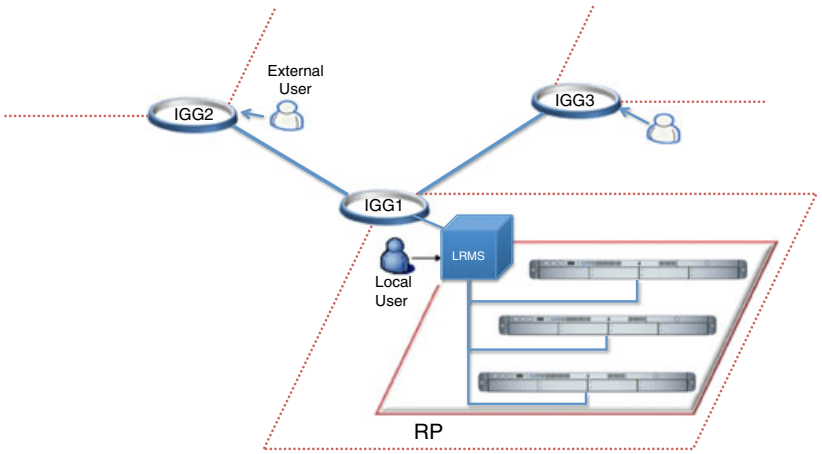


Figure 9. Evaluation scenario based on three InterGrid Gateways (IGGs). RP, resource provider; LRMS, local resource manager system.

Table II. Characteristics of lease requests used in the experiments.

| Request ID | Arrival time | No. nodes | Memory (MB) | Duration (s) | Type |
|---|---|---|---|---|---|
| 1 | 00:00:00 | 3 | 256 | 3600 | External |
| 2 | 00:05:00 | 1 | 128 | 5400 | External |
| 3 | 00:06:00 | 2 | 128 | 5400 | External |
| 4 | 00:08:00 | 1 | 256 | 5400 | External |
| 5 | 00:08:50 | 2 | 64 | 2400 | External |
| 6 | 00:09:40 | 3 | 128 | 3600 | External |
| 7 | 00:12:00 | 5 | 128 | 3600 | Local |

ID, identification.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

For evaluation, we first demonstrate how our implementation enables InterGrid to resolve the contention between local and external requests. It shows the effect of preempting existing external leases on a virtualized physical testbed to satisfy the requirements of an arriving local request. For that purpose, we compare the baseline situation, that is, when there is not any preemption policy (No Policy (NOP)) against situation that the MOML preemption policy is applied. In the former, the local request (request ID: 7) is rejected; whereas in the latter, external leases (request identification: five and six) are preempted and vacate resources for the local requests. We notice that the local request is served without being delayed. Additionally, the RP could utilize its resources more efficiently by leasing them to the external requests.

More specifically, Figure 10 indicates how the MOML contention resolution policy allocates resources to the local request in comparison with NOP. In fact, the vertical axis in this figure shows the resource utilization variations, whereas the horizontal axis presents the overall makespan time to run the workload.

In the beginning, the resource utilization rapidly increases to 100% for both policies because of allocating resources to the arriving external requests (requests one to six in Table II). As time passes, we observe that the resource utilization gradually drops to 0% as the requests are completed. We can see that the resource utilization reduction is sharper for NOP than MOML. Indeed, when the resources are 100% utilized and the local request arrives, the MOML policy preempts external requests and schedules them after the local request. After completing the local request, the preempted external requests are resumed. As a result, the MOML policy operates with higher utilization for longer time to run the local request. Additionally, as it is presented in Figure 10, preempting external request in favor of the local requests and resuming them at a later time leads to longer makespan time for the MOML policy.

Various contention resolution policies (preemption policies) preempt different leases. Therefore, they lead to different amount of resource contention and overhead time. In the next experiment, we evaluate the efficacy of the implemented policies from the overhead, overall makespan time, and resource contention aspects. Specifically, we measure how many resource contentions are resulted from different policies. Additionally, we work out how much overhead imposed to the system by suspension and resumption operations on the preempted leases.

In Table III, the number of resource contentions as well as the amount of overhead resulted from MLIP, MOV, and MOML policies are listed. As we can see, the MLIP policy affects two leases, and the overall size of memory should be written/read to/from memory is 1152 MB. The MOV policy that aims at minimizing the overall preemption overhead. Therefore, it preempts leases that impose minimum overhead to the system (i.e. $\{5, 2, 3\}$), and the amount of memory should be deallocated and snapshot on the disk is 512 MB that implies 27.8 s overhead. MOML affects just two leases ($\{5, 6\}$), whereas results in 512 MB of memory suspension and resumption overhead.

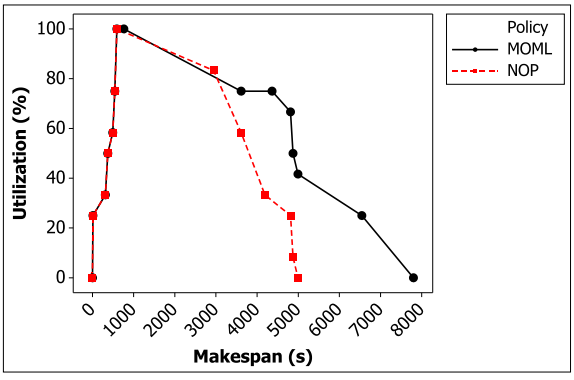


Figure 10. Efficacy of the preemption mechanism by comparing minimum overhead minimum lease (MOML) preemption policy against baseline situation that there is no contention resolution in place (NOP). The vertical axis shows how the resource utilization varies over the time using these policies. The horizontal axis shows the overall makespan of running the requests in seconds.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Table III. Number of resource contentions (lease preemption), overhead, and makespan resulted from applying different preemption policies.

| Policy | Preempted leases | Overhead (s) | Makespan (s) |
|---|---|---|---|
| MLIP | {1, 6} | 59.9 | 7800 |
| MOV | {5, 2, 3} | 27.8 | 8479 |
| MOML | {5, 6} | 27.8 | 7800 |

MLIP, minimum leases involved policy; MOV, minimum overhead; MOML, minimum overhead minimum lease.

The comparison of the results from different policies indicates that the selection of different preemption policies is effective on the number of contentions and time overhead imposed to the system.

## 9. CONCLUSION AND FUTURE WORK

In this work, we explored how the resource contention between local and external requests can be resolved in a virtualized large-scale distributed system. For this purpose, we applied a preemption-based approach where external leases are preempted in favor of local requests. We observed that preempting leases substantially decrease the rejection of local requests (up to 72% with 95% confidence interval:(51.1, 92.8)) without increasing external requests rejection rate. Furthermore, we investigated the side effects of preemption mechanism when VMs are used for resource provisioning. Specifically, we modeled the overhead of suspending and migrating operations on VMs. The advantage of the proposed model is considering the possible communications between VMs at the time of preemption.

Then, we considered three policies to decide which lease(s) are better choices for preemption. The MOV policy that aims at reducing the imposed overhead and improving resource utilization. The MLIP increases user satisfaction. However, it does not provide a good resource utilization. The MOML policy makes a trade-off between resource utilization and user satisfaction.

Although the problem we are investigating in this paper is in the InterGrid context, it could be also applied to other lease-based Grid/Cloud RPs where requests with higher priority (such as local or organizational requests) coexist with other requests. For instance, in Amazon EC2 datacenters, different types of instances with distinct priorities coexist, for example, Spot, On-demand, and Reserved instances. However, there are some differences between the Cloud computing and the environment we considered in this research. Most importantly, Cloud computing environments are generally economy driven, thus, require cost-aware preemption policies.

In the future, we plan to extend the current work by considering economy-based preemption policies. Another interesting future direction is situation where there is a dependency between leases. Furthermore, we are interested in scenarios where local requests are also from different types (e.g., local suspendable and local migratable).

ACKNOWLEDGEMENTS

This work is partially supported by research grants from the Australian Research Council (ARC). We would like to thank Ehsan Nadjaran Toosi for his help in improving algorithms' performances. This paper is a substantially extended version of our conference paper presented at Australian Computer Science Conference in 2011 (ACSC '11) [11].

REFERENCES

1. Hermenier F, Lèbre A, Menaud J. Cluster-wide context switch of virtualized jobs. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*, USA, 2010; 658–666.
2. Walters JP, Bantwal B, Chaudhary V. Enabling interactive jobs in virtualized data centers. *Cloud Computing and Applications* 2008; **1**:21–26.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

3. Sotomayor B, Montero RS, Llorente IM, Foster I. Resource leasing and the art of suspending virtual machines. *Proceedings of the 11th IEEE International Conference on High Performance Computing and Communications*, USA, 2009; 59–68.
4. Bolze R, Cappello F, Caron E, Daydé M, Desprez F, Jeannot E, Jégou Y, Lanteri S, Leduc J, Melab N, Mornet G, Namyst R, Primet P, Quetier B, Richard O, El-Ghazali T, Touche I. Grid'5000: a large scale and highly reconfigurable experimental Grid testbed. *International Journal of High Performance Computing Applications* 2006; **20**(4):481–497.
5. Iosup A, Epema DHJ, Tannenbaum T, Farrellee M, Livny M. Inter-operating grids through delegated matchmaking. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '07)*. ACM: USA, 2007; 1–12.
6. De Assunção MD, Buyya R, Venugopal S. InterGrid: a case for internetworking islands of Grids. *Concurrency and Computation: Practice and Experience* 2008; **20**(8):997–1024.
7. Grit L, Ramakrishnan L, Chase J. On the duality of jobs and leases. *Technical Report CS-2007-00*, Duke University, Department of Computer Science, April 2007.
8. Sotomayor B, Keahey K, Foster I. Combining batch execution and leasing using virtual machines. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*. ACM: USA, 2008; 87–96.
9. Chase JS, Irwin DE, Grit LE, Moore JD, Sprenkle SE. Dynamic virtual clusters in a Grid site manager. *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, USA, 2003; 90–98.
10. Zhao M, Figueiredo RJ. Experimental study of virtual machine migration in support of reservation of cluster resources. In *Proceedings of the 3rd International Workshop on Virtualization Technology in Distributed Computing*. ACM: Barcelona, 2007; 5–11.
11. Amini Salehi M, Javadi B, Buyya R. Resource provisioning based on leases preemption in InterGrid. In *Proceeding of the 34th Australasian Computer Science Conference (ACSC'11)*, Vol. 113, CRPIT. ACS: Perth, Australia, 2011; 25–34.
12. Snell Q, Clement MJ, Jackson DB. Preemption based backfill. In *Job Scheduling Strategies for Parallel Processing (JSSPP '02)*. Springer: Edinburgh, 2002; 24–37.
13. Tsafrir D, Etsion Y, Feitelson DG. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems* 2007; **18**(6):789–803.
14. Lawson BG, Smirni E. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. *ACM SIGMETRICS Performance Evaluation Review* 2002; **29**(4):40–47.
15. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M. A view of cloud computing. *Communications of the ACM* 2010; **53**(4):50–58.
16. Kettimuthu R, Subramani V, Srinivasan S, Gopalsamy T, Panda DK, Sadayappan P. Selective preemption strategies for parallel job scheduling. *International Journal of High Performance and Networking (IJHPCN)* 2005; **3**(2/3):122–152.
17. Isard M, Prabhakaran V, Currey J, Wieder U, Talwar K, Goldberg A. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd Symposium On Operating Systems Principles (SOSP '09)*. ACM: Big Sky, 2009; 261–276.
18. Fontán J, Vázquez T, Gonzalez L, Montero RS, Llorente IM. OpenNEbula: the open source virtual machine manager for cluster computing. *Open Source Grid and Cluster Software Conference*, USA, 2008.
19. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D. The Eucalyptus open-source cloud-computing system. *1st Workshop of Cloud Computing and Its Applications*, Chicago, Vol. 1, 2008; 124–131.
20. Vanmechelen K, Depoorter W, Broeckhove J. Combining futures and spot markets: a hybrid market approach to economic grid resource management. *Journal of Grid Computing* 2011; **9**(1):81–94.
21. Iosup A, Sonmez O, Anoep S, Epema D. The performance of bags-of-tasks in large-scale distributed systems. *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, USA, 2008; 97–108.
22. Buyya R, Murshed MM, Abramson D, Venugopal S. Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimization algorithm. *Software: Practice and Experience* 2005; **35**(5):491–512.
23. Costanzo Ad, Jin C, Varela CA, Buyya R. Enabling computational steering with an asynchronous-iterative computation framework. *Proceedings of the 5th IEEE International Conference on e-Science*, USA, 2009; 255–262.
24. Kwok Y, Ahmad I. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Transaction Parallel and Distributed Systems* 1996; **7**(5):506–521.
25. Sharma B, Thulasiram RK, Thulasiraman P, Garg SK, Buyya R. Pricing cloud compute commodities: a novel financial economic model. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '12. IEEE Computer Society: Ottawa, 2012; 451–457.
26. Fallenbeck N, Picht HJ, Smith M, Freisleben B. Xen and the art of cluster scheduling. In *Proceedings of the 1st International Workshop on Virtualization Technology in Distributed Computing (VTDC '06)*. IEEE: Tampa, 2006; 4–12.
27. Sulistio A, Cibej U, Venugopal S, Robic B, Buyya R. A toolkit for modelling and simulating data grids: an extension to GridSim. *Concurrency and Computation: Practice and Experience (CCPE)* 2008; **20**(13):1591–1609.


Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

28. Lublin U, Feitelson DG. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing* 2001; **63**:1105–1122.
29. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. KVM: the Linux virtual machine monitor. *Linux Symposium*, Ottawa, 2007; 225–232.
30. Salehi MA, Krishna PR, Deepak KS, Buyya R. Preemption-aware energy management in virtualized datacenters. *Proceeding of 5th International Conference on Cloud Computing (IEEE Cloud '12)*, Hawaii, USA, 2012; 844–851.
31. Noronha R, Panda DK. IMCa: a high performance caching front-end for GlusterFS on InfiniBand. *Proceedings of the 37th International Conference on Parallel Processing (ICPP '08)*, USA, 2008; 159–165.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe