SOFTWARE – PRACTICE AND EXPERIENCE
Softw. Pract. Exper. (2013)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2221

Contention management in federated virtualized distributed systems: implementation and evaluation

Mohsen Amini Salehi[1, 2,*,†], Adel Nadjaran Toosi[2] and Rajkumar Buyya[2]

[1]*Electrical and Computer Engineering Department, Colorado State University, Fort Collins, CO, U.S.A.*
[2]*Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia*

SUMMARY

The paper describes creation of a contention-aware environment in a large-scale distributed system where the contention occurs to access resources between external and local requests. To resolve the contention, we propose and implement a preemption mechanism in the InterGrid platform, which is a platform for large-scale distributed system and uses virtual machines for resource provisioning. The implemented mechanism enables the resource providers to increase their resource utilization through contributing resources to the InterGrid platform without delaying their local users. The paper also evaluates the impact of applying various policies for preempting user requests. These policies affect resource contention, average waiting time, and imposed overhead to the system. Experiments conducted in real settings demonstrate efficacy of the preemption mechanism in resolving resource contention and the influence of preemption policies on the amount of imposed overhead and average waiting time. Copyright © 2013 John Wiley & Sons, Ltd.

Received 1 October 2012; Revised 17 June 2013; Accepted 9 August 2013

KEY WORDS: resource contention; preemption mechanism; lease-based resource provisioning; virtual machine (VM)

1. INTRODUCTION

Federated large-scale distributed systems [1], such as InterGrid, enable sharing, selection, and aggregation of resources across several resource providers (RPs). These RPs (e.g., clusters or sites) are usually connected through high-bandwidth network connections. Nowadays, heavy computational requirements, mostly from scientific communities [2], are supplied by these RPs.

The main objective of InterGrid is to provide a software platform for interconnecting disjoint RPs. The RPs motivation for contributing resources to a resource sharing environment is to increase their utilization and revenue. However, they would like to ensure that the requirements of local (organizational) requests are met. Therefore, resource provisioning within such RPs is carried out for two types of user requests, namely, local and external requests. Local requests are submitted by local users to the local resource management system (LRMS) of the RP, whereas external requests are submitted by users in other RPs that tend to access larger amount of resources through InterGrid.

The mixture of local and external requests within RPs leads to a resource contention between requests to access resources. Typically, in this situation, local requests have priority over external requests [3]. Indeed, the organization that owns the resources welcomes external requests to use resources if they are available. Nonetheless, the external requests should not delay the execution of local requests.

*Correspondence to: Mohsen Amini Salehi, Electrical and Computer Engineering Department, Colorado State University, Fort Collins, CO, U.S.A.

†E-mail: amini@engr.colostate.edu, mohsena@csse.unimelb.edu.au

Similar to many current distributed systems, InterGrid [4] provides resources in the form of VM-based leases. 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 [5]. Virtual machine (VM) technology is a suitable vehicle for implementing lease-based provisioning model [5]. VM capabilities in getting suspended, resumed, stopped, or even migrated (when there is enough bandwidth) have been extensively studied and shown to be useful in resource provisioning without major utilization loss [3, 5, 6]. InterGrid creates one lease for each user request. Leases in InterGrid are contiguous and must be served within resources of a single RP.

The current implementation of the InterGrid platform does not recognize the resource contention between local and external requests and treats them equally. In fact, the current implementation of InterGrid suffers from lack of proper resource management at the RP level that considers the difference between local and external requests. Therefore, in this work, we extended the InterGrid platform by introducing LRMS at the RP level that recognizes the resource contention by differentiating local and external requests. The LRMS employs preemption mechanism to resolve the contention between local and external requests. That is, when there is not enough vacant resources to be allocated to an arriving local request, external leases are preempted and relinquish resources for the arriving local request. The preempted external leases are scheduled at a later time in the same RP to resume their execution.

Preempting VM-based leases entails the overhead time of suspending VMs of that lease (i.e., unloading memory contents of the VMs into the disk) and resuming them (i.e., loading VMs from the disk image into the memory) at the scheduled time. Thus, the amount of imposed overhead time for preemption depends on the VM characteristics, such as memory size, in that lease. In addition, preempting external leases and scheduling them at a later time increase the waiting time of external leases.

For a local request that requires multiple VMs, possibly, several external leases have to be preempted to make sufficient vacant resources. Therefore, there are potentially several sets of candidate external leases that can be preempted. In this paper, each set of the candidate leases is termed a *candidate set*. Choosing different candidate sets for preemption affects the amount of imposed overhead and the average waiting time of external leases.

These issues raise another problem that is choosing the optimal candidate set for preemption in a way that minimizes the side effects of preempting leases (i.e., overhead and waiting time). Therefore, as another contribution of this paper, we evaluate the impact of different preemption policies on the side effects of preemption.

In summary, this paper makes the following contributions:

1. It designs and implements LRMS component for the InterGrid platform (Section 3.2).
2. It recognizes resource contention between local and external requests in the InterGrid and resolves it by preempting external leases in favor of arriving local requests (Section 4.3).
3. It implements different preemption policies that choose different candidate sets for preemption (Section 4.3.1).
4. It evaluates the impact of the proposed policies upon key performance metrics in a real environment (Section 5.2).

This work describes design and implementation details of the contention-aware scheduling in the InterGrid platform. More specifically, the work realizes the architecture proposed in Section 3 and explains a system that enables provisioning of VM-based leases for different types of users. Although the problem we investigate in this paper is in the InterGrid context, it can be applied to other lease-based grid/Cloud RPs where requests with higher priority (such as local or organizational requests) coexist with other requests.

The remainder of this paper is organized as follows. In the next section, we provide an overview of the related work. Then, in Section 3, the InterGrid architecture is described. After that, in Section 4, we explain the design and implementation details. Performance evaluations are mentioned in Section 5. Finally, conclusion and future works are provided in Section 6.

DOI: 10.1002/spe

2. RELATED WORKS

As presented in Figure 1, different mechanisms that have been employed in interconnected distributed systems to resolve the resource contention are preemption, partitioning, and using multiple queues.

Preemption mechanism, which is also the mechanism we employ in this research, stops the running request and free resources for another, possibly higher priority or urgent, request. If checkpointing is supported in a system (e.g., [7]), then the preempted request can resume its execution from the preempted point. Otherwise, the preempted request has to be killed (canceled) or restarted its execution. For parallel requests (i.e., those need more than one processing element at the same time), *full preemption* is generally performed, in which the whole request leaves the resources. However, some systems support *partial preemption* (e.g., [8]), in which a portion of resources allocated to a parallel request is preempted.

In the *partitioning* mechanism, resources are reserved for different request types (e.g., local and external requests). Boundaries of the reservations (partitions) can be fixed (static partitioning) or can be flexible on the basis of the demand of each request type (dynamic partitioning). For instance, in [9], the local and external partitions can borrow resources from the other one when there is a high demand of either.

In the *multiple queues* mechanism (e.g., [10]), requests are classified in distinct queues upon arrival. Each queue operates independently and has its own policy for the requests within the queue. There is a higher level scheduling policy that determines the appropriate queue for dispatching a request to the resources.

In systems that function on the basis of VMs, preemption mechanism is preferable to resolve resource contention, because it removes the burden of partitioning or managing several queues from a resource management system. Although preemption mechanism has not been studied extensively in the distributed computing systems previously [11], dominance of the VM technology as the provisioning model has led to undertake several studies in this area [12–15]. In this section, we review recent studies that have applied preemption mechanism to resolve the contention between requests.

NDDE [16] is a platform that utilizes VMs to exploit idle cycles for grid or cluster usage in corporations and educational institutions. This system is able to utilize idle cycles that appear even while the user is interacting with the computer. In this system, the guest and the owner applications are executed concurrently. This approach increases the harvested idle cycle to as many as possible with minor impact on the interactive user's applications. NDDE has more priority than the idle process in the host operating system, therefore, it is executed instead of the Idle process. When a new request from the resource owner arrives, the VM and all the processes that belong to NDDE are preempted and changed to 'ready-to-run' state.

Cluster-on-demand (COD) [17] is a resource management system for shared clusters. COD supports lease-based resource provisioning in the form of virtual clusters where each virtual cluster is an isolated group of hosts inside a shared hardware base. It is equipped with a protocol that dynamically resizes virtual clusters in cooperation with middleware components. It uses group-based priority and partial preemption scheme to manage request-initiated resource contention. Specifically, when resource contention takes place, COD preempts nodes from a low-priority virtual cluster. For preemption, the selected virtual cluster returns those nodes that create minimal disruption to the virtual clusters (i.e., an instance of partial preemption).

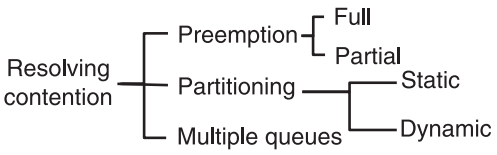

Figure 1. Approaches for resolving resource contention in interconnected distributed systems.

DOI: 10.1002/spe

VioCluster [18], is a VM-based platform across several clusters. It uses lending and borrowing policies to trade VMs between clusters. VioCluster is equipped with a machine broker that decides when to borrow/lend VMs from/to another cluster. Machine broker also has policies for reclaiming resources by preempting a leased VM to another domain. Machine property policy monitors the machine properties that should be allocated to the VMs such as CPU, memory, and storage capacity. Location policy in the VioCluster proactively determines if it is better to borrow VMs from another cluster or to wait for nodes on a single domain.

Snell *et al*. [11] applied preemption on the backfilling scheduling strategy. They provide policies to select the set of requests for preemption in a way that the requests with higher priority are satisfied and, at the same time, the resource utilization increases. The preempted request is restarted and rescheduled in the next available time slot.

Scojo-PECT [19] provides a limited response time for several job classes within a virtualized Cluster. They propose a coarse-grained preemption and suspending VMs on the disk to cope with the resource contention. The preemptive scheduler aims at creating a fair share scheduling between different job classes of a grid.

A preemptive scheduling algorithm is implemented in MOSIX [20] to allocate excess (unclaimed) resources to users that require more resources than their share. However, these resources will be released as soon as they are reclaimed. MOSIX also supports the situations that there are local and guest jobs and considers a priority between them.

3. INTERGRID ARCHITECTURE

The architecture of InterGrid, as illustrated in Figure 2, presumes that each grid is composed of several RPs. RPs can be in the form of a cluster, symmetric multiprocessor or a combination of these resources. Each RP is managed independently and has its own local users while contributing resources to InterGrid. The LRMS of an RP handles resource provisioning for local and external requests within that RP. It is worth noting that the current implementation if InterGrid suffers from lack of LRMS within RPs and creation of this component is one of the contributions of this paper.

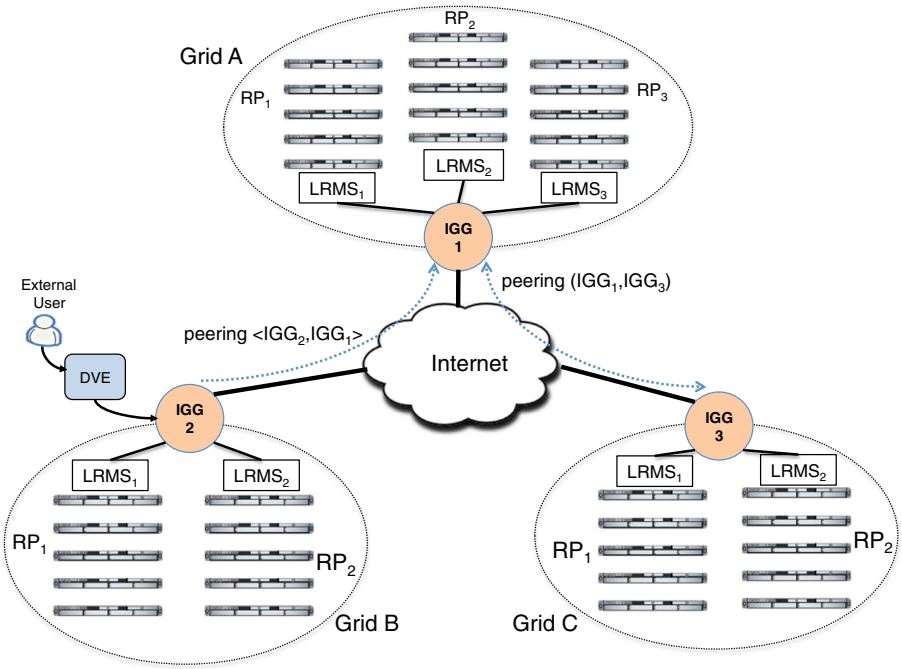

Figure 2. High-level architecture of InterGrid. Grid B has unilateral peering arrangement, and grid C has bilateral peering arrangement with grid A.

DOI: 10.1002/spe

Utilization and revenue enhancement are the main motivations for RPs to contribute resources to InterGrid. However, external requests should not delay the execution of local requests. Therefore, in this work, we consider external requests as best effort, whereas local requests should not be delayed and have to be served within their requested interval. Best-effort allocation means that if there is no sufficient resources to start an external request, it is scheduled in the earliest available time slot. However, in situation that there is no adequate resources for an arriving local request, LRMS should preempt external requests and makes the resources available for the arriving local request. It is worth noting that if all resources are occupied by other local requests, or the amount of resources released by preempting external requests are not sufficient for the local request, then the arriving local request is rejected.

In the InterGrid platform, sharing resources amongst different grids is achieved through predefined arrangements that are known as peering arrangements (see Figure 2). Such arrangement was initially inspired from principles of the Internet policy-based routing. Similar idea is applied to interconnection of grids to offload and redirecting resource requests from one grid to another [21]. Peering arrangement between two grids coordinates adoption of resources between the two grids based on policies on how resources are allocated to users of peering grids. For instance, in Figure 2, grid *B* has chosen a unidirectional peering arrangement with grid *A*. In this case, grid *B* can redirect resource requests to grid *A*, whereas the opposite is not possible. Peering between grids are formed on the basis of monetary contracts between these grids (e.g., certain amount of resources from a particular grid during a month).

Peering arrangements with other grids are handled by InterGrid gateways (IGG). An IGG is aware of the peering terms between the grids. When a user requires resources from other grids, IGG chooses the suitable grid that can provide the required resources based on the peering arrangements with other grids. An IGG also replies to resource requests from other peer IGGs through allocation of resources to them. In fact, provisioning rights over RPs within a grid are delegated to the IGG that enable it to schedule arriving external requests on the RPs.

Distributed virtual environment manager (DVE manager) is a user level tool in the InterGrid architecture. Users willing to access InterGrid level resources (i.e., external users) utilize this tool to interact with IGG and acquire resources. The DVE manager also handles monitoring of the resources that are allocated to the user.

3.1. InterGrid gateway structure

InterGrid gateway is the core part of the InterGrid platform that has been implemented in Java. A high-level view of the IGG components is depicted in Figure 3.

The main component of the IGG structure is the *scheduler* that implements provisioning policies and peering with other IGGs. Specifically, the *peering* part of the scheduler selects an appropriate peer in order to redirect a request to another grid. The *provisioning* part of the scheduler in an IGG maps arriving external requests to the available RPs within the grid. Provisioning decisions of the scheduler are enacted on the RPs through the virtual machine manager interface.

Generic implementation of the virtual machine manager enables an IGG to interact with diverse platforms used as LRMS in the RPs. Therefore, to enable an IGG to interact with a new platform in an RP, the interface for interacting with that platform has to be developed. So far, several interfaces have been developed in IGG. As it is shown in Figure 3, these interfaces have enabled IGG to interact with OpenNebula [22] and Eucalyptus [23] to manage the resources in RPs. Moreover, interfaces have been developed for IGG that enable it to interact with Grid'5000 as a grid middleware and Amazon EC2[‡] as a cloud infrastructure as a service provider. In addition to this, an emulated interface has been developed for testing and debugging purposes.

In Figure 3, the *persistence* database is used for storing information needed by IGG, such as information of peering arrangements with other grids. The *management and monitoring* module provides command-line tools to configure and manage an IGG. This module has been implemented on the

[‡]http://aws.amazon.com/ec2

DOI: 10.1002/spe

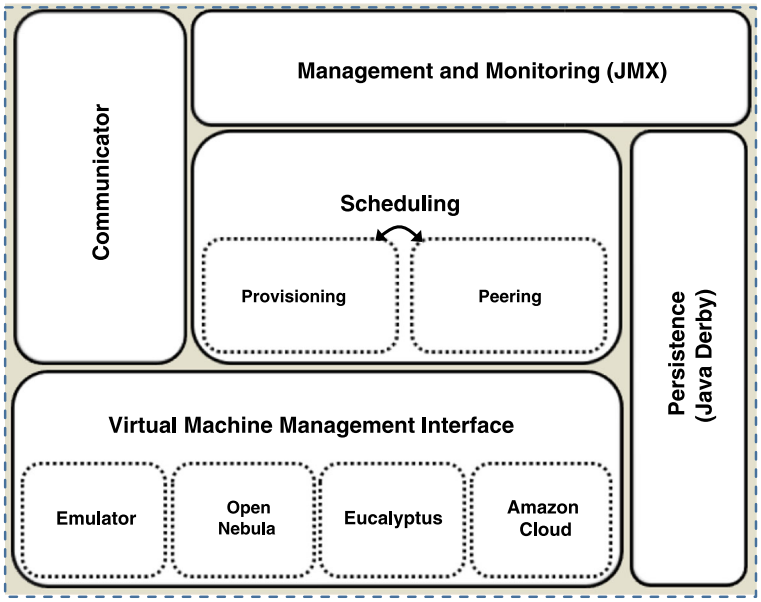

Figure 3. Components of the InterGrid gateway.

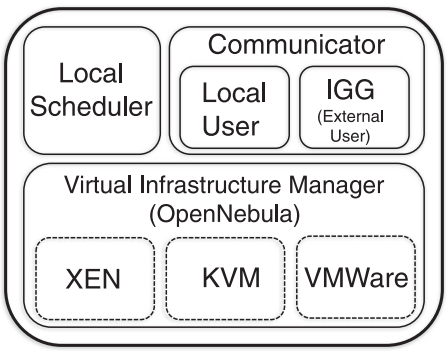

Figure 4. Architectural view of local resource management system that is used within each resource provider.

basis of Java Management eXtensions technology. The *communication* module provides an asynchronous message-passing mechanism between IGGs that makes IGGs loosely coupled and fault tolerant.

3.2. Local resource management system

Because of the flexible architecture of IGG, each RP in a grid can have its own local management platform. In this paper, the local management platforms of RPs are called LRMS. The current implementation of InterGrid does not have LRMS, and implementation of this component is one of the contributions of this paper.

According to Figure 4, an LRMS is mainly comprised of a *virtual infrastructure manager (VIM)* (e.g., OpenNebula [22]) that performs operations such as creating, starting, and stopping VMs. VIM enacts its operations through hypervisors (e.g., Xen, KVM) installed on the worker nodes of an RP. A *local scheduler* determines how resources are allocated to the arriving local or external request on the RP's nodes. The scheduler considers VM as the provisioning unit and is able to preempt external

DOI: 10.1002/spe

leases in favor of an arriving local request. The local scheduler employs aggressive backfilling as the scheduling strategy of RP's resources.

The *communicator* module is comprised of a part that receives lease requests from local users and another part that interacts with IGG. The latter is in charge of advertising and updating availability information of the RP on the IGG. In addition, it receives external lease requests from the IGG. Any received request is delivered to the scheduling module of the LRMS.

4. SYSTEM DESIGN AND IMPLEMENTATION

This section provides details on how we implemented the contention-aware scheduling within the InterGrid platform. In this regard, first, the model we proposed for resource allocation of local and external requests in InterGrid is described. Then, design and implementation details of the proposed resource allocation model is discussed.

4.1. Design of contention-aware resource allocation model

In this part, we describe resource acquisition steps for a user request in InterGrid. The steps are different for external and local users' requests. The workflow for resource acquisition for external requests in InterGrid is illustrated in Figure 5 and is described as follows.

1. An RP advertises its resources availabilities periodically in the registry of an IGG. This advertisement also implies delegation of the provisioning rights of the RP's resources to the IGG.
2. Using a DVE manager, an external user initiates a lease request by specifying details of its request. A lease request describes required resources to be deployed and has the following characteristics:

    - Virtual machine template (describes specifications of the requested VMs).
    - Number of VMs.
    - Lease duration (also known as lease wall time).
    - Lease deadline (this is optional for external requests).

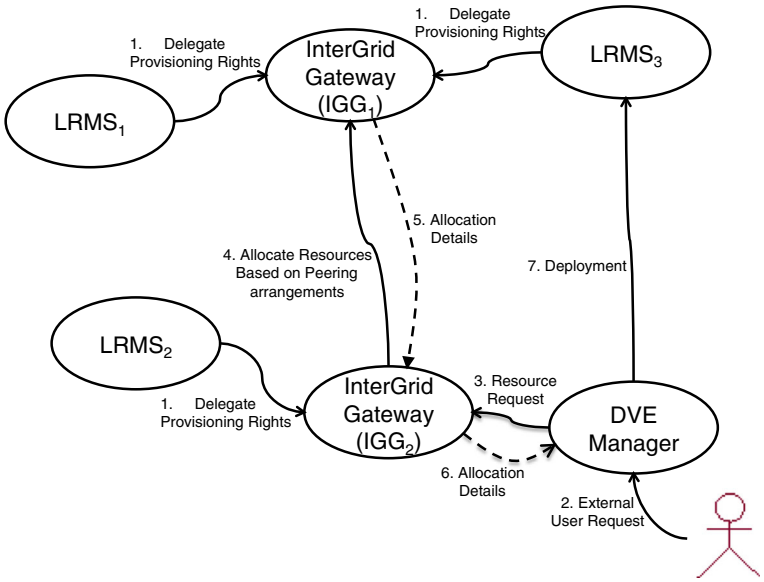

Figure 5. Resource allocation steps for external user requests in InterGrid.

DOI: 10.1002/spe

3. The DVE manager forms a lease request and passes that to its correspondent IGG to acquire resources through InterGrid.
4. Provisioning unit of the IGG scheduler first tries to schedule the lease request on RPs within its grid. However, if the grid cannot provide the required resources, possibly because of oversubscription, then the IGG uses its peering arrangements with other grids (e.g., IGG_1 in Figure 5) to provide resources for the lease request. It is worth mentioning that the peering policy decides which peering grid should be selected amongst several candidates grids.
5. Once the peering grid (e.g., IGG_1 in Figure 5) receives a lease request, its provisioning unit allocates resources for the lease request on its RPs. Then, the requestor IGG (e.g., IGG_2 in Figure 5) is informed about the allocation details.
6. Requestor DVE manager is given a permission and other deployment information (e.g., IP address) to deploy the VMs from the allocated RP at the scheduled time.
7. At the scheduled time, the DVE manager is connected to the LRMS of the destination RP and deploys the requested VMs.

Resource acquisition procedure for local users of an RP is as follows.

1. Local users send lease requests to the LRMS of their RP. This is achieved using any user-level interface provided by the LRMS of an RP.
2. Local resource management system tries to serve (i.e., schedule) local lease requests during their requested interval using its available resources.
3. If resources are free during the requested interval, then resource allocation is performed.
4. If during this interval resources are occupied by other local requests (i.e., contention with other local leases), then the lease cannot be allocated and it will be rejected. In this situation, the local user may resubmit its lease request to InterGrid in order to access resources elsewhere for the requested duration.
5. If resources of an RP are occupied by external requests during the requested interval (contention with external leases), then LRMS preempts some or all of external leases in order to serve the local request. The preempted external lease is scheduled in the next available time slot.

4.2. Implementation of resource allocation for external requests

Resource allocation workflow for external requests that was described in Figure 5 is implemented within IGG. The sequence diagram of invoking the IGG classes to accept and allocate an external request is shown in Figure 6. For the sake of clarity, this figure just shows parts of the whole provisioning process that take place within IGG.

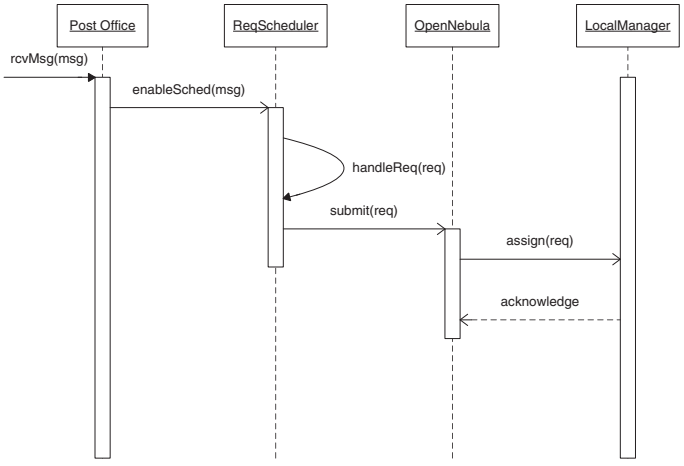

Figure 6. Scheduling external user requests in IGG.

DOI: 10.1002/spe

When a message is received by an IGG, it is handled by a central component, called *post office*, which spawn one thread to handle each message. In the case that the received message is an external request received through peering arrangement with another IGG, the spawned thread invokes the scheduler in the *request scheduler* class. At this stage, the scheduling is carried out by invoking *handleRequest* method that extracts the request from the message and allocates it to an appropriate RP. The scheduling of external requests on an appropriate RP is carried out in a proactive manner with the aim of reducing the number of contentions within RPs. Interested readers can refer to [24] for more details on the rationale of the scheduling methods within IGG. Allocation of the scheduled external request starts with calling *submit* method that is a method in virtual machine management interface (see Figure 3). This method is implemented by different classes that implement the virtual machine management interface—OpenNebula in our setting. In the last step, the OpenNebula interface in IGG calls *assign* method to send the external request to the LRMS of the selected RP. The way an external request is handled in the LRMS of an RP is explained in the next subsection.

4.3. Implementation of contention-aware scheduling within LRMS

The critical part of a contention-aware LRMS is a *scheduler* that allocates resources across an RP. The local scheduler should be aware of the contention between local and external requests within an RP.

Another component of LRMS is a virtual infrastructure manager that offers an on-demand provisioning model in form of VMs (we employed OpenNebula [22] as the virtual infrastructure manager that is explained in Section 4.3.2). However, on-demand provisioning of VMs is not sufficient for resource provisioning in InterGrid. Specifically, lease requests in InterGrid have the following requirements:

- Priority between local and external users.
- Capacity reservation at specified times for local requests.
- Preemption throughout VMs' lifetime.

The local scheduler that operates on top of the virtual infrastructure manager has to cover these requirements. Haizea is an open source scheduler developed by Sotomayor *et al*. [25] that employs VM-based leases for resource provisioning. The advantage of Haizea is enabling VM preemption (i.e., suspending and resuming) and considering overheads of deploying and preempting VMs in the scheduling.

We adopt and customize Haizea to be the local scheduler component of LRMS in RPs. This scheduler operates on top of virtual infrastructure manager and enables recognition of the contention between local and external requests that occurs within an RP. More importantly, adopting Haizea as the local scheduler enables leasing resources to external requests in a best-effort manner while allocating resources to the local requests within their requested time interval. When a resource contention occurs (i.e., a local request arrives and resources are occupied by external leases), the local scheduler resolves it through preempting the external lease(s) and vacating resources to serve the local request. In this way, the local scheduler operates as the scheduling back-end of virtual infrastructure manager (i.e., OpenNebula). The local scheduler also employs aggressive backfilling scheduling strategy [11] along with VMs' abilities (i.e., suspend and resume) to efficiently schedule leases and to increase resource utilization.

The sequence diagram of invocations between the local scheduler classes is shown in Figure 7. The scheduling process in the local scheduler starts by receiving a lease request either from local or external user (through IGG) in the *LocalManager* class.

The manager requests the *LeaseScheduler* class to schedule the lease request. Then, the *schedule* method in the *VMScheduler* class is called, which schedules local and external requests. For local requests, VMs are scheduled on the basis of the requested time interval. External requests are allocated in the first vacant space. The *map* function in the *mapper* class maps requested resources to the physical resources based on their availability times. When the mapper class handles a local request,

DOI: 10.1002/spe

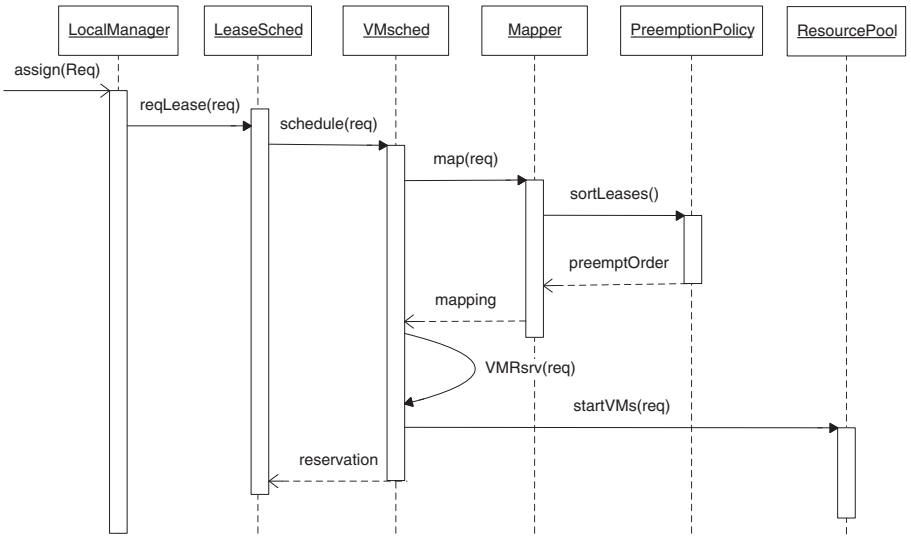

Figure 7. Scheduling local requests in the local scheduler.

if there is not enough resources, then the mapper calls the *PreemptionPolicy* to determine the preferred order of preempting external leases. The order is determined on the basis of the preemption policy. That is, the preemption policy determines the proper set of external leases to be preempted from a set of possible external leases that their preemption creates enough room for allocation of an arriving local request. More details on preemption policies can be found in Section 4.3.1. In the next step, in Figure 7, the *mapper* performs the mapping and returns the mapping list to the *VMScheduler*.

Using the mapping information, VMScheduler calls the *VMRsrv* and updates the scheduling information of the resources. After that, the lease can be started by calling the *startVMs* method in the *ResourcePool* class. In addition, the *LeaseScheduler* is informed to update all of the affected leases in the scheduling table.

4.3.1. Preemption policies. The local scheduler described in the previous part enables recognition of the contention between local and external requests and resolves it using the preemption technique. However, preemption technique implies side effects that should be taken into account. Overhead imposed for suspending and resuming VMs is one side effect of the preemption technique. The second side effect is the waiting time of preempted leases. In fact, preempted leases are rescheduled in a later time slot, which increases the overall waiting time of the leases.

Time overhead of preemption is a factor that the resource owner wants to minimize (i.e., instance of a system-centric metric), whereas minimum waiting time (MWT) of external request is a factor that end users expect from the system (i.e, a user-centric metric). However, administrator of an RP may want to make a balance between these factors. Therefore, a combinatorial policy is also needed to consider both of these factors.

In this work, we have proposed and implemented the following preemption policies in the local scheduler that proactively detect the resource contentions and try to minimize their side effects.

- Minimum waiting time. This policy tries to minimize the average waiting time of external lease requests through selecting leases for preemption that has MWT. For this purpose, leases with the MWT are given higher preemptibility score. In the event that all requests have the same waiting time, ties are broken by selecting younger leases. That is, leases that have been submitted more recently receive higher preemptibility score. This policy disregards the time overhead imposed for preempting leases.

DOI: 10.1002/spe

- Minimum overhead (MOV). This policy tries to minimize the overall overhead time imposed to the system by preempting VMs. Implementation of the policy is based on selecting set of leases for preemption that result in the MOV time. For that purpose, we calculate the overhead imposed by suspending and resuming a lease with n VMs as follows [26].

$$score = \frac{1}{n \cdot \left(\frac{mem}{s} + \frac{mem}{r}\right)} \tag{1}$$

  where mem is the amount of memory requested by VMs of a lease, s is the rate of transferring megabytes of VM memory to disk per second, and r is the rate of reallocating megabytes of VM memory per second. Similar to MWT, in the event that all requests have the same waiting time, ties are broken by selecting younger leases.
- Combinatorial preemption policy (CP). This policy considers both waiting time and overhead in selecting leases for preemption. In this policy, preemptibility score is calculated on the basis of Equation (2), where v indicates the amount of overhead imposed by preempting a lease based on Equation (2), and w is the waiting time of the lease.

$$score = \alpha \cdot v + (1 - \alpha) \cdot w \tag{2}$$

  Performance of this policy depends on the value of α. The current value of α is worked out experimentally through sweeping different values for that. Details of optimally determining the value of α is described in Section 5.1.

4.3.2. Virtual infrastructure manager. We utilize OpenNebula [22] as the virtual infrastructure manager to handle VMs' lifecycle across an RP. OpenNebula provides a software layer between a service and hypervisor and enables dynamic provisioning of resources to services.

The architecture of OpenNebula has been designed to be flexible and modular to be able to support various hypervisors and infrastructure configurations within an RP (e.g., cluster). It enables dynamic resource partitioning and provides web and command-line interfaces that allows local users to conveniently request leases. For each user request, OpenNebula starts, monitors, and stops VMs according to the provisioning policies in place. OpenNebula architecture includes three main elements as follows:

- Core is responsible for managing VMs' life cycle by performing basic operations such as start, migrate, monitor, and terminate [27].
- Capacity manager consists of pluggable policies that determine the VM placement across an RP. Default capacity manager in OpenNebula provides a simple VM placement and load balancing policies. In particular, it uses an *on-demand* provisioning model, where virtualized resources are allocated at the time they are requested, without the possibility of requesting resources at a specific future interval.
- Virtualizer access drivers provide the abstraction for the underlying virtualization layer by exposing the general functionalities of the hypervisor (e.g., start, migrate, and terminate). As a result of this component, OpenNebula is able to work with various hypervisors such as Xen, KVM, and VMware.

OpenNebula is equipped with databases for storing VM templates [22]. A template file consists of a set of attributes that defines a VM, such as the number of processing elements, amount of memory, and disk. OpenNebula uses a shared storage to store VM images. The shared storage model requires the head node (where LRMS and OpenNebula reside) and hosts (where hypervisors and VMs reside) to share the VM image repository. Typically, these storage areas are shared using a distributed file system such as NFS [28] and GlusterFS [29]. A shared storage reduces VM deployment times and enables live migration, but it can also become a bottleneck in the infrastructure and degrade the VMs performance specially for executing disk-intensive workloads. We used GlusterFS as the shared storage in our configuration.

DOI: 10.1002/spe

For the following reasons, we used GlusterFS.

- It provides a scalable, shared replicated storage across hosts over Ethernet.
- Compared with other centralized shared storage approaches, such as NAS, GlusterFS does not suffer from single point of failure. That is, due to its distributed nature, even if a host fails the storage and data will remain available.
- GlusterFS is highly scalable, as I/O operations are distributed across hosts.
- As data are replicated, VMs have local data access that remarkably improves the I/O performance.

4.3.3. Virtualization infrastructure. Along with the virtual infrastructure manager, a virtualization infrastructure (i.e., a hypervisor) is required in each host of an RP to manage VMs. Specifically, utilizing OpenNebula as a virtual infrastructure manager enables us to deploy various hypervisors within an RP.

In our implementation, we use Kernel-based virtual machine (KVM) [30] as the hypervisor within each host of the RPs. KVM is a hardware-assisted, fully virtualized platform for Linux on X86 hardware that has virtualization extensions. By installing KVM, multiple execution environments (guest VMs from different disk images) can be created on top of physical machines. Each of these VMs has private and virtualized hardware that includes a network card, storage, memory, and graphics adapter.

5. PERFORMANCE EVALUATION

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

- A four-node cluster as the RP. Hosts are four IBM Systems X3200 M3 machines, each with a quad-core Intel Xeon x3400, 2.7 GHz processor with a 4 GB memory. The head node, where LRMS resides, is a Dell Optiplex 755 machine with Intel Core 2 Duo E4500, 2.2 GHz processor and 2 GB memory.
- The hosts' operating system is CentOS 6.2 Linux distribution. The operating system in the head node is Ubuntu 12.4.
- All the nodes are connected through a 100-Mbps switched Ethernet network.
- The RP's LRMS is comprised of OpenNebula 3.4, as the virtual infrastructure manager, and Haizea version 1.1, as the local scheduler.
- Qemu-KVM 0.12.1.2 was installed as the hypervisor on each host.
- GlusterFS is used as the cluster file system. It aggregates commodity storages across a cluster and forms a large parallel network file system [29]. 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 in our experiment involves an InterGrid with three IGGs with peering arrangements established between them, as illustrated in Figure 8. IGG1 has the cluster as the RP and users from IGG2 and IGG3 request external leases through their DVE managers. On the basis of the peering arrangements, IGG1 provides resources for requestors. IGG1 receives these requests in the form of external requests, and they are allocated resources through LRMS of the RP. However, the RP has its own local requests that have more priority than the external ones.

To have a realistic evaluation, the experiments are carried out on the basis of real traces from the blue horizon cluster [31] in San Diego Supercomputer Center. We consider the first 60 requests in this workload as the external requests received by IGG1. Local lease requests received by LRMS of IGG1 are explained in the Table I.

According to the table, 13 local lease requests are submitted to the RP. Each row of the table shows the arrival time, number of requested processing elements, amount of memory, start time, and the duration of local requests. The reason that we consider few requests is that all times are in real. Also, we wanted to demonstrate the feasibility of the implementation and be able to trace the

DOI: 10.1002/spe

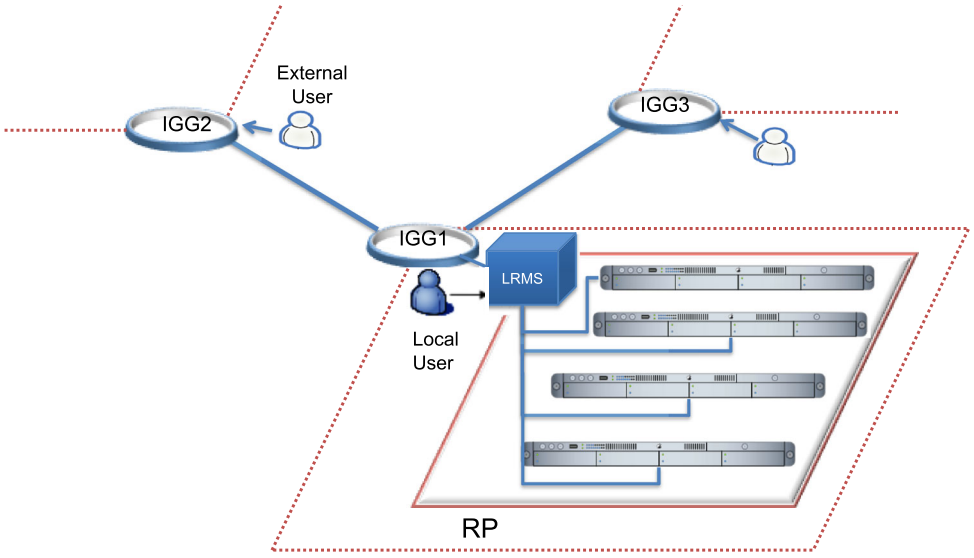

Figure 8. Evaluation scenario based on three InterGrid Gateways.

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

Request ID	Arrival time	No. nodes	Memory (MB)	Start time	Duration (s)
1	01:00:00	4	1024	01:15:00	00:45:00
2	02:00:00	4	2048	02:15:00	01:00:00
3	03:00:00	8	512	03:15:00	02:40:00
4	04:00:00	4	1024	04:15:00	00:10:00
5	05:00:00	2	1024	05:15:00	00:35:00
6	06:00:00	8	2048	06:15:00	01:12:00
7	07:00:00	8	1024	07:15:00	02:10:00
8	08:00:00	4	512	08:15:00	00:32:00
9	09:00:00	4	1024	09:15:00	01:00:00
10	10:00:00	4	2048	10:15:00	03:10:00
11	11:00:00	4	1024	11:15:00	01:00:00
12	12:00:00	4	2048	12:15:00	02:35:00
13	13:00:00	8	1024	13:15:00	01:40:00

order of events taking place in the system. We consider 00:00:00 as the start of the experiment, 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.

5.1. Managing trade-off between waiting time and overhead

To implement the CP policy, we need to determine the value of α. In fact, this value determines the behavior of this policy. System administrator can use this parameter to adjust behavior of the system. According to Equation (2), if $\alpha = 0$, then the policy becomes MWT, whereas $\alpha = 1$ transforms the CP policy to the MOV policy.

In order to determine an appropriate value for α, we swept 100 possible values of α (i.e., $0 \leqslant \alpha \leqslant 1$ with $step = 0.01$), and the result is shown in Figure 9. Each point in this figure shows the overhead, and waiting time resulted from applying a specific value of α in the CP policy.

Figure 9 shows that the behavior of the CP policy does not change entirely linearly by changing α. In other words, there are points where we can compromise a little bit of one parameter and receive a huge gain on the other parameter. Nonetheless, selecting a proper value for α is defined on the basis of the administrative policies in an RP. For instance, if the administrative policy is giving more weight to user-centric metrics (i.e., lower average waiting time), the lower values on the waiting

DOI: 10.1002/spe

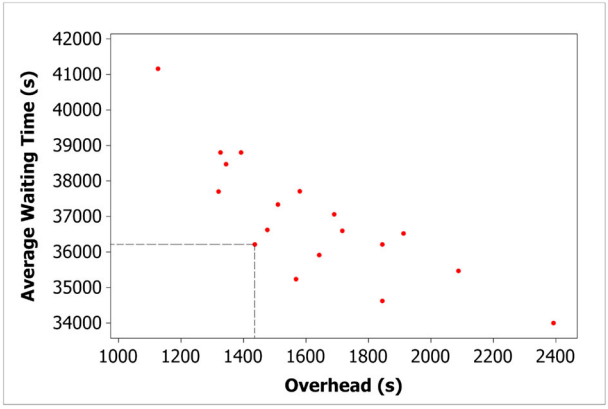

Figure 9. Variations in the overhead and average waiting time when α parameter in Equation (2) changes from 0 to 1.

time axes should be selected that results into a high overhead. In our case, we selected values of $\alpha = 0.31$ (the dashed point in Figure 9). We can see that at this point, a huge gain on the average waiting time can be attained by compromising a little bit on the overhead.

5.2. *Evaluation results*

In the first experiment, we 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 only compare situation that there is not any preemption policy (NOP) against situation that the CP preemption policy is applied. In the former, all the local requests were rejected, whereas in the latter, when there is no sufficient resources, external leases are preempted and vacate resources for the local requests. We notice that all of the local requests are served without being delayed. In addition, the RP could utilize its resources more efficiently by leasing them to the external requests.

Various contention resolution policies (preemption policies) preempt different leases. These policies lead to different amount of overhead and average waiting time. In the second experiment, we evaluate the efficacy of the implemented policies from the overhead, average waiting time, and overall makespan time aspects. The overhead imposed to the system is imposed by suspension and resumption operations on the preempted leases. Calculation of suspension and resumption operations are worked out on the basis of the read/write throughput of our Gluster file system, which is 40 MB/s [29].

In Table II, the amount of overhead, makespan, and average waiting time resulted from MWT, MOV, and CP policies that are shown. As we can see, the MWT policy results in less waiting time and more overhead time compare with MOV. The MOV policy aims at minimizing the overall preemption overhead. Therefore, it preempts leases that impose MOV to the system and based on the amount of memory should be de-allocated and snapshot on the disk.

Table II. Total imposed overhead, average waiting time, and makespan results from applying different preemption policies.

Policy	Overhead (s)	Average waiting time (s)	Makespan (s)
MWT	1912	36520.33	190,665
MOV	1028	39984.44	195,971
CP	1476	36621.6.03	194,686

DOI: 10.1002/spe

We can also see that CP can make a trade-off between overhead (as a system-centric metric) and waiting time (as a user-centric metric). In other words, it results in a good waiting time and makespan while imposing overhead just a little bit more than MOV.

The comparison of the results from different policies indicates that the selection of different preemption policies is effective on the side effects of resource contention. In addition, CP policy can be used by a system administrator to manage the trade-off between user-centric and system-centric metrics in an RP based on the administrative policies and priorities.

6. CONCLUSION

We presented the real implementation and evaluation of the preemption mechanism in InterGrid where local and external leases coexist in RPs. We also designed, implemented, and incorporated LRMS component of the InterGrid platform that adheres to the flexible and diverse nature of large-scale distributed systems. The system prototype demonstrates how an RP can contribute resources to InterGrid and accepting external requests without affecting the local requests. The provided implementation also implements several preemption policies for the RPs. Evaluation of the preemption policies indicated how the average waiting time and overall makespan time and amount of imposed overhead are affected by these policies. Specifically, such policies can be used by the system administrator to make a trade-off between diverse parameters based on administrative policies of an RP.

REFERENCES

1. Iosup A, Epema DHJ, Tannenbaum T, Farrellee M, Livny M. Inter-operating grids through delegated matchmaking. *Proceedings of the ACM/IEEE Conference on Supercomputing*, SC '07, ACM, New York, NY, USA, 2007; 13:1–13:12.
2. Buyya R, Murshed M, 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.
3. 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*, Washington, DC, USA, 2003; 90–98.
4. De Assunção M, Buyya R, Venugopal S. InterGrid: a case for internetworking islands of Grids. *Concurrency and Computation: Practice and Experience* 2008; **20**(8):997–1024.
5. Sotomayor B, Keahey K, Foster I. Combining batch execution and leasing using virtual machines. *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, ACM, New York, NY, USA, 2008; 87–96.
6. Zhao M, Figueiredo R. Experimental study of virtual machine migration in support of reservation of cluster resources. *Proceedings of the 3rd International Workshop on Virtualization Technology in Distributed Computing*, Spain, 2007; 5–11.
7. Coti C, Herault T, Lemarinier P, Pilard L, Rezmerita A, Rodriguez E, Cappello F. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant mpi. *Proceedings of the ACM/IEEE Conference on Supercomputing*, USA, 2006; 17–132.
8. Sandholm T, Lai K, Clearwater S. Admission control in a computational market. *8th IEEE International Symposium on Cluster Computing and the Grid*, France, 2008; 277–286.
9. Assunção MD, Buyya R. Performance analysis of multiple site resource provisioning: effects of the precision of availability information. *Proceedings of the 15th International Conference on High Performance Computing,* HiPC '08, India, 2008; 157–168.
10. Lawson B, Smirni E. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. *ACM SIGMETRICS Performance Evaluation Review* 2002; **29**(4):40–47.
11. Snell Q, Clement MJ, Jackson DB. Preemption based backfill. In *Job Scheduling Strategies for Parallel Processing (JSSPP)*. Springer: Heidelberg Germany, 2002; 24–37.
12. Amini Salehi M, Javadi B, Buyya R. Resource provisioning based on leases preemption in InterGrid. *Proceeding of the 34th Australasian Computer Science Conference (ACSC'11)*, Perth, Australia, 2011; 25–34.
13. Amini Salehi M, Javadi B, Buyya R. Preemption-aware admission control in a virtualized grid federation. *Proceeding of 26th International Conference on Advanced Information Networking and Applications (AINA'12)*, Japan, 2012; 854–861.
14. Amini Salehi M, Javadi B, Buyya R. Performance analysis of preemption-aware scheduling in multi-cluster grid environments. *Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'11)*, Australia, 2011; 419–432.

DOI: 10.1002/spe

15. Amini Salehi M, Krishna PR, Deepak KS, Buyya R. Preemption-aware energy management in virtualized datacenters. *Proceeding of 5th International Conference on Cloud Computing (IEEE Coud'12)*, USA, 2012.
16. Novaes R, Roisenberg P, Scheer R, Northfleet C, Jornada J, Cirne W. Non-dedicated distributed environment: a solution for safe and continuous exploitation of idle cycles. *Scalable Computing: Practice and Experience* 2001; **6**(3):107–115.
17. Moore J, Irwin D, Grit L, Sprenkle S, Chase J. Managing mixed-use clusters with cluster-on-demand. *Tech. rep.*, Duke University Department of Computer Science, 2002.
18. Ruth P, McGachey P, Xu D. VioCluster: virtualization for dynamic computational domain. *IEEE International on Cluster Computing (Cluster'05)*, Burlington, USA, 2005; 1–10.
19. Sodan A. Service control with the preemptive parallel job scheduler scojo-pect. *Journal of Cluster Computing* 2010; **14**(2):1–18.
20. Amar L, Mu'Alem A, Stober J. The power of preemption in economic online markets. *Proceedings of the 5th International Workshop on Grid Economics and Business Models*, GECON '08, Berlin, Heidelberg, 2008; 41–57.
21. De Assunção M, Buyya R. Performance analysis of allocation policies for intergrid resource provisioning. *Information and Software Technology* 2009; **51**(1):42 –55.
22. 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 – Book of Abstracts*, San Francisco, USA, 2008.
23. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D. The eucalyptus open-source cloud-computing system. *9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09)*, China, 2009; 124–131.
24. Amini Salehi M, Javadi B, Buyya R. Qos and preemption aware scheduling in federated and virtualized grid computing environments. *Journal of Parallel and Distributed Computing (JPDC)* 2012; **72**(2):231–245.
25. 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*, Washington, DC, USA, 2009; 59–68.
26. 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.
27. Sotomayor B, Montero RS, Llorente IM, Foster I. Capacity leasing in cloud systems using the opennebula engine. *Workshop on Cloud Computing and its Applications*, Amazon, 2008; 3(2).
28. Network file system. Available from: http://www.ibm.com/developerworks/linux/library/l-nfsv4/index.html/ [last accessed April 2013].
29. 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, Washington, USA, 2008; 462–469.
30. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. KVM: the linux virtual machine monitor. *Linux symposium*, Canada, 2007; 225–232.
31. Parallel workloads archive. Available from: http://www.cs.huji.ac.il/labs/parallel/workload/ [last accessed March 2013].

DOI: 10.1002/spe