57

Taxonomy of Contention Management in Interconnected Distributed Systems

Mohsen Amini Salehi
The University of Melbourne

Jemal Abawajy
Deakin University

Rajkumar Buyya
The University of Melbourne

57.1 Introduction

Scientists and practitioners are increasingly reliant on large amounts of computational resources to solve complicated problems and obtain results in a timely manner. To satisfy the demand for large computational resources, organizations build or utilize distributed computing systems.

A distributed computing system is essentially a set of computers that share their resources via a computer network and interact with each other toward achieving a common goal [31]. The shared resources in a distributed system include data, computational power, and storage capacity. The common goal can also range from running resource-intensive applications, tolerating faults in a server, and serving scalable Internet applications.

Distributed computing systems such as Clusters, Grids, and recently Clouds have become ubiquitous platforms for supporting resource-intensive and scalable applications. However, surge in demand is still a common problem in distributed systems [26] in a way that no single system (specially Clusters and Grids) can meet the needs of all users. Therefore, the notion of interconnected distributed computing systems has emerged.

In an interconnected distributed computing system, as depicted in Figure 57.1, organizations share their resources over the Internet and consequently are able to access larger resources. In fact,

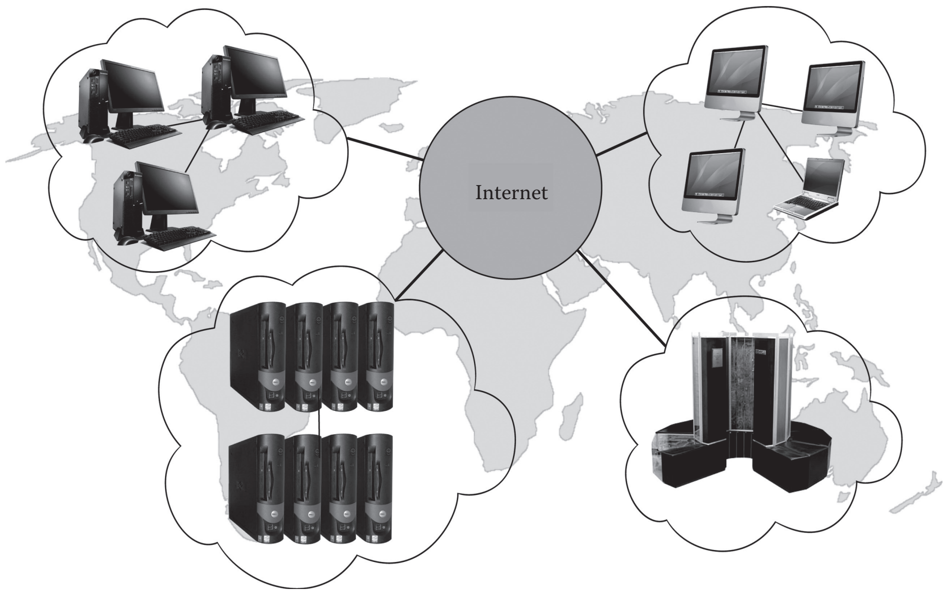

FIGURE 57.1 Interconnected distributed computing systems.

interconnected distributed systems construct an overlay network on top of the Internet to facilitate resource sharing between the constituents.

However, there are concerns in interconnected distributed systems regarding contention between requests to access resources, low access level, security, and reliability. These concerns necessitate a resource management platform that encompasses these aspects. The way current platforms consider these concerns depends on the structure of the interconnected distributed system. In practice, interconnection of distributed systems can be achieved in different levels. These approaches are categorized in Figure 57.2 and explained over the following paragraphs.

- User level (Broker-based): Is useful for creating loosely coupled interconnected distributed systems. In this approach, users/organizations are interconnected through accessing multiple distributed systems. This approach involves repetitive efforts to develop interfaces for different distributed systems and, thus, scaling to many distributed systems is difficult. Gridway [103] and GridBus broker [104] are examples of broker-based interconnection approach. The former, achieves interconnection in organization level, whereas the latter, works in the enduser level.
- Resource level: In this approach, different interfaces are developed on the resource side and consequently the resource can be available to multiple distributed systems. This approach involves administration overhead, since the resource administrator has to be aware of well-known services. This approach is difficult to scale to many distributed systems, hence, it is suggested mostly for large distributed systems. Interconnection of EGEE, NorduGrid, and D-Grid is done based on

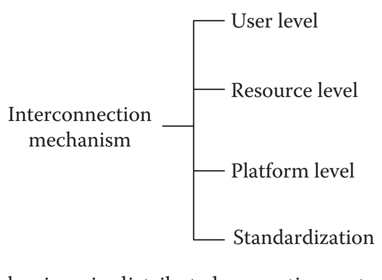

FIGURE 57.2 Interconnection mechanisms in distributed computing systems.

this approach [31]. Particularly, D-Grid [35] leverages interconnectivity via implementing interfaces of UNICORE, gLite, and Globus on each resource provider in a way that resources can be accessed by any of the middlewares.

- Platform level (Gateways): A third platform (usually called a gateway) handles the arrangements between distributed systems. Ideally, the gateway is transparent both from users and resources and makes the illusion of single system for the user. However, in this approach gateways are single point of failure and also a scalability bottleneck. InterGrid [26] and the interconnection of Naregi and EGEE [65] are instances of this approach.
- Standardization: Common and standard interfaces have been accepted as a comprehensive and sustainable solution for interconnecting distributed systems. However, current distributed systems (e.g., current Grid platforms) have already been developed based on different standards and it is a hard and long process to change them to a common standard interface. Issues regarding creating standards for interconnecting distributed systems are also known as interoperability of distributed systems.

  UniGrid [88] is a large-scale interconnected distributed system implemented based on a standard and connects more than 30 sites in Taiwan. It offers a web interface that bridges the user and the lower-level middleware. The core of UniGrid orchestrates different middlewares, including Globus Toolkit [33], Condor [96], and Ganglia [80] transparently from the user. Another project that sought to achieve the idea of World Wide Grid through developing standards and service-oriented architecture is GRIP [25].

Grid computing is a prominent example of interconnected distributed systems. Grids are usually comprised of various organizations that share their resources (e.g., Clusters or SMPs) and form Virtual Organizations (VOs). The concept of Grid has specifically been fascinating for users/organizations that did not have huge resources available or did not have the budget to manage such resources. Nowadays, Grids are utilized predominantly in scientific communities to run high performance computing (HPC) applications. Over the last decade, variety of Grids have emerged based on different interconnection mechanisms. TeraGrid in the United States [102], DAS in the Netherlands [61], and Grid5000 in France [17] are such examples.

Generally, in an interconnected environment, requests from different sources co-exist and, therefore, these systems are prone to contention between different requests competing to access resources. There are various types of contentions that can occur in an interconnected distributed system and, accordingly, there are different ways to cope with these contentions.

The survey will help people in the research community and industry to understand the potential benefits of contention-aware resource management systems in distributed systems. For people unfamiliar with the field, it provides a general overview, as well as detailed case studies.

The rest of this chapter is organized as follows: In Section 57.2, an overview on resource management systems of interconnected distributed systems is presented. Next, in Section 57.3 contention in interconnected distributed systems is discussed which is followed by investigating the architectural models of the contention-aware resource management systems in Section 57.4. In Section 57.5, we discuss about different approaches for contention management in well-known interconnected distributed systems. Finally, conclusion and avenues of future works for researchers are provided in Section 57.6.

57.2 Request Management Systems

Interconnected distributed systems, normally, encounter various users and usage scenarios from users. For instance, the following usage scenarios are expectable:

- Scientists in a research organization run scientific simulations, which are in the form of long running batch jobs without specific deadlines.
- A corporate web site needs to be hosted for a long period of time with a guaranteed availability and low latency.
- A college instructor requires few resources at certain times every week for demonstration purposes.

In response to such diverse demands, interconnected distributed systems offer different service levels (also called multiple quality of service (QoS) levels).

For example, Amazon EC2* supports reserved (availability guaranteed), on-demand, and spot (best-effort) virtual machine (VM) instances. Offering a combination of advance-reservation and best-effort schemes [93], interactive and batch jobs [109], tight-deadline and loose-deadline jobs [37] are common practices in interconnected distributed systems.

These diverse service levels usually imply different prices and priorities for the services that have to be managed by the resource management system. Additionally, interconnected distributed systems can be aware of the origin of the requests and they may discriminate requests based on that. Another challenge in job management of interconnected distributed systems is managing accounting issues of sending/receiving requests to/from peer distributed systems.

There are many approaches for tackling these challenges in resource management systems of interconnected distributed systems. One common approach is prioritizing requests based on criteria, such as service (QoS) or origin. For example, in an interconnected distributed system usually local requests (i.e., local organizations' users) have priority over the requests from external users [5]. Another example is in urgent computing [15] (urgent applications), such as earthquake and bush-fire prediction applications where the applications intend to acquire many resources in an urgent manner. In circumstances that there is surge in demand, requests with different priorities compete to gain access to resources. This condition is generally known as resource contention between requests.

Resource contention is the main challenge in request management of interconnected distributed systems and occurs when a user request cannot be admitted or cannot receive adequate resources, because the resources are occupied by other (higher priority) requests.

In the remainder of this survey, we explore different aspects of resource contention in interconnected distributed systems and also we investigate the possible solutions for them.

57.3 Origins of Resource Contentions

There are various causes for resource contention in interconnected distributed systems. They broadly can be categorized as request-initiated, interdomain-initiated, origin-initiated and hybrid. A taxonomy of different contention types along with their solutions is shown in Figure 57.3.

57.3.1 Request-Initiated Resource Contention

Request-initiated resource contention occurs if any of the requests monopolizes resources to such an extent that deprives other requests from gaining access to them. It is prevalent in all forms of distributed systems, even where there is no interconnection. There are several scenarios that can potentially lead to request-initiated resource contention. One important situation is when there is an imbalance in request sizes, mainly, in terms of required number of nodes or execution time (duration). In this circumstance, small requests may have to wait for a long time behind a long job to access resources.

Another cause for request-initiated resource contention is situation that requests have QoS constraints and they selfishly try to satisfy them. Generally, resource management systems can support three types of QoS requirements for users' requests:

1. Hard QoS: Where the QoS constraints cannot be negotiated. These systems are prone to QoS violation and, hence, managing resource contention is critical [73].
2. Soft QoS: Where the QoS constraints are flexible and can be negotiated based upon the resource availabilities or when there is a surge in demand. The flexibility enables resource management systems to apply diverse resource contention solutions [73].

* http://aws.amazon.com/ec2/

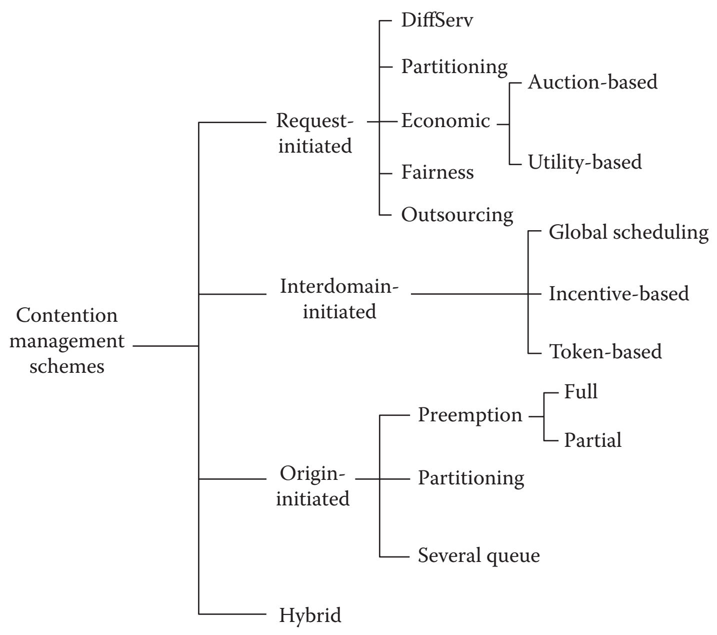

FIGURE 57.3 Taxonomy of different types of resource contentions and possible solutions in interconnected distributed computing systems.

3. Hybrid QoS: Where the resource management system supports a combination of Hard QoS and Soft QoS requirements for the user requests. This fashion is common in commercial resource providers such as Cloud providers. For instance, Amazon EC2 supports services with distinct QoS requirements including reserved (hard QoS), and spot (soft QoS) VM instances. Another example, are the resource management systems that support combination of interactive (hard QoS) and batch requests (usually soft QoS) [109].

Solutions for managing request-initiated contentions are mostly achieved in the context of scheduling and/or admission control units of resource management systems. Over the next paragraphs, we categorize and describe different solutions for resource contention.

Differentiated services (DiffServ) technique that initially was used in Computer Networks and developed to guarantee different QoS levels (with different priorities) for various Internet services, such as VOIP and web. In Computer Networks, DiffServ guarantees different QoSs through dividing the services into distinct QoS levels. According to IETF RFC 2474, each level is supported by dropping TCP packets of lower priority levels.

Similar approach can be taken in the context of request-initiated resource contentions in distributed systems. For this purpose, the resource management system presents different QoS levels for user requests. Then, requests are classified in one of these levels at the admission time. However, in this scheme there is no control on the number of requests assigned to each QoS level. As a result, QoS requirements of request cannot be guaranteed. Therefore, DiffServ scheme is appropriate for soft QoS requirements.

Variations of DiffServ technique can be applied when contention occurs due to imbalanced requests. Silberstein et al. [89] also sought to decrease the response time of short requests in a multigrid environment. For that purpose, they apply a multilevel feedback queue (MLFQ) scheduling. In their policy, Grids are placed in different categories based on their response speed. Requests are all sent to the first queue upon arrival and if they cannot get completed in the time limit of that level, then they are migrated to the lower level queue which is a larger grid. The process continues up until the task finishes or reaches down the hierarchy.

In the *Partitioning scheme*, the resources are reserved for requests with different QoS levels. Unlike DiffServ scheme, in this approach boundaries of the reservations (partitions) can adaptively move, based on the demand in different QoS levels. This solution can also be considered as a type of DiffServ that is suitable for requests with hard QoS requirements.

Economic scheme solutions either work in an auction-based or utility-based manner. In the former, both resource provider and resource consumer have their own agents. Through an auctioneer the consumer bids on the resources and also provides a valuation function. Then, the provider agent tries to maximize the utility based on the valuation function and comes up with a set of resources for the user. In the latter, a utility function that generally reflects the revenue earned by running a request is calculated for all contentious requests. Then, the request that maximizes the utility function has the priority of accessing resources. These approaches are commonly applied in market-oriented scheduling [36].

Fair scheme that guarantees contentious requests receive their share of the system resources [3]. This scheme is used to resolve resource contentions resulting from imbalanced requests in the system and assures starvation-free scheduling of the requests.

Outsourcing scheme: Interconnection of distributed systems creates the opportunity to employ resources from other distributed systems in the case of resource contention. Outsourcing is applied for both causes of request-initiated resource contention (i.e., request imbalance and QoS levels). Specially, Cloud providers have been extensively employed for outsourcing requests [84]. This issue has helped in emergence of hybrid clouds, which are a combination of a private (organizational) resources and public Clouds [12]. Although we categorize outsourcing as a resolution for request-initiated contentions, it can be applied for interdomain and origin initiated contentions as will be discussed in the next parts.

57.3.2 Interdomain-Initiated Resource Contention

Interdomain-initiated resource contention occurs, when the proportion of shared resources to the consumed resources by a constituent distributed system is low. In other words, this resource contention happens when a resource provider contributes few resources while demand more resources from other resource providers in an interconnected distributed system. Unlike request-initiated contention, which merely roots in request characteristics and can take place in any distributed system, interdomain contention is based on the overall consumption and contribution of each resource provider.

There are several approaches for handling interdomain-initiated contentions, namely, global scheduling, incentive, and token-based schemes (see Figure 57.3). These approaches are discussed in detail in what follows .

Global schedulers: In this approach, which is appropriate for large-scale distributed systems, there are local (domain) schedulers and global (meta) schedulers. Global schedulers are in charge of routing user requests to local schedulers and, ultimately, local schedulers, such as Condor [96] or Sun Grid Engine (SGE) [16], allocate resources to the requests.

Global schedulers can manage the interdomain resource contention by admitting requests from different organizations based on the number of requests it has redirected to the resources of each organization. Since global schedulers usually are not aware of the instantaneous load condition in the local schedulers, it is difficult for them to guarantee QoS requirements of users [11]. Thus, this approach is useful for circumstances where requests have soft QoS requirements.

Incentive scheme: In this approach, which is mostly used in peer-to-peer systems [71], resource providers are encouraged to share resources to be able to access more resources. Reputation Index Scheme [58] is a type of incentive-based approach in which the organization cannot submit requests to another organization while it has less reputation than that organization. Therefore, in order to gain reputation, organizations are motivated to contribute more resources to the interdomain sharing environment.

Quality service incentive scheme [70] is a famous type of incentive-based approach. Quality service is an extension of Reputation Index Scheme. The difference is that depending on the number of QoS levels offered by a participant, a set of distinct ratings is presented where each level has its own reputation index.

Token-based scheme: Operates based on the principle where a certain amount of tokens, which are allocated to an organization, is proportional to its resource contribution. If a user wants to get access to another organization resources, its consumer agent must spend amount of tokens to get the access. This scheme encompasses request-initiated and interdomain resource contentions. To address the request-initiated resource contention, valuation functions can be used to translate the QoS demands of user to the number of tokens to be used for a request. The provider agent can then use its own valuation functions to compute the admission price for the request. Finally, the request will be admitted only if the admission price is less or equal to the number of tokens that the requesting organization is willing to pay [73].

57.3.3 Origin-Initiated Resource Contention

In interconnected distributed systems, users' requests originate from distinct organizations. More importantly, these systems are prone to resource contention between local requests of the organization and requests from other organizations (i.e., external requests). Typically, local requests of each organization have priority over external requests [5]. In other words, the organization that owns the resources would like to ensure that its community has priority access to the resources. Under such a circumstance, external requests are welcome to use resources if they are available. Nonetheless, external requests should not delay the execution of local requests.

In fact, origin-initiated resource contention is a specific case of interdomain-initiated and request-initiated resource contentions. Consequently, the approaches of tackling this type of resource contention is similar to the already mentioned approaches. Particularly, partitioning approach both in static and dynamic forms and global scheduling are applicable for origin-initiated resource contentions. There are also other approaches to cope with origin-initiated contentions that we discuss in this part.

Preemption scheme: This mechanism stops the running request and free the resources for another, possibly higher priority, or urgent request. The higher priority request can be a local request or a hard QoS request in an interconnected distributed system. The preempted request may be able to resume its execution from the preempted point. If suspension is not supported in a system, then the preempted request can be killed (canceled) or restarted. For parallel requests, *full preemption* usually is performed, in which whole request leaves the resources. However, some systems support *partial preemption*, in which part of resources allocated to a parallel request is preempted [86].

Although preemption mechanism is a common solution for origin-initiated contentions, it is also widely applied to solve request-initiated resource contentions. Due to the prominent role of preemption in resolving these types of resource contentions, in Section 57.4.5 we explain preemption in details.

Partitioning scheme: Both static and dynamic partitioning of resources, as mentioned in Section 57.3.1, can be applied to tackle origin-initiated contentions.

In dynamic partitioning of resources, the local and external partitions can borrow resources from each other when there is a high demand of local or external requests [11].

Several queues: In this approach when requests arrive [59], they are categorized in distinct queues, based on their origin. Each queue can independently have its own scheduling policy. Then, another scheduling policy determines the appropriate queue that can dispatch a request to the resources.

Combinations of the aforementioned contentions (mentioned as hybrid in Figure 57.3) can occur in an interconnected distributed system. The most common combination is the origin-initiated and request-initiated resource contentions. For instance, in federated Grids and federated Clouds, origin-initiated contention occurs between local and external requests. At the same time, external and local requests can also have distinct QoS levels, which is a request-initiated resource contention [5,6,81]. Generally, Resolution of hybrid resource contentions is a combination of different strategies mentioned earlier.

57.4 Contention Management

Resource management system is the main component of a distributed system that is responsible for resolving resource contentions. Various elements of a resource management system contribute in resolving different types of resource contentions. They apply different approaches in managing contentions. Different components of resource management systems and the way they deal with resource contention is presented in Figure 57.4.

57.4.1 Resource Provisioning

Resource provisioning component of a resource management system is in charge of procuring resources based on user application requirements. Resource provisioning is performed based on a provisioning model that defines the execution unit in a system. In fact, requests are allocated resources based on the resource provisioning model.

Resource provisioning models do not directly deal with resource contentions. However, the way other components of resource management system function strongly depends on the resource provisioning model.

Provisioning resources for users' requests in distributed systems has three dimensions as follows:

1. Hardware resources
2. Software available on the resources
3. Time during which the resources are available (availability)

Satisfying all of these dimensions in a resource provisioning model has been challenging. In practice, past resource provisioning models in distributed systems were unable to fulfill all of these dimensions [93]. Emergence of virtual machine (VM) technology as a resource provisioning model recently has posed an opportunity to address all of these dimensions. Over the next subsections, we discuss common resource provisioning models in current distributed systems.

57.4.1.1 Job Model

In this model, jobs are pushed or pulled across different schedulers in the system to reach the destination node, where they can run. In job-based systems, scheduling a job is the consequence of a request to run

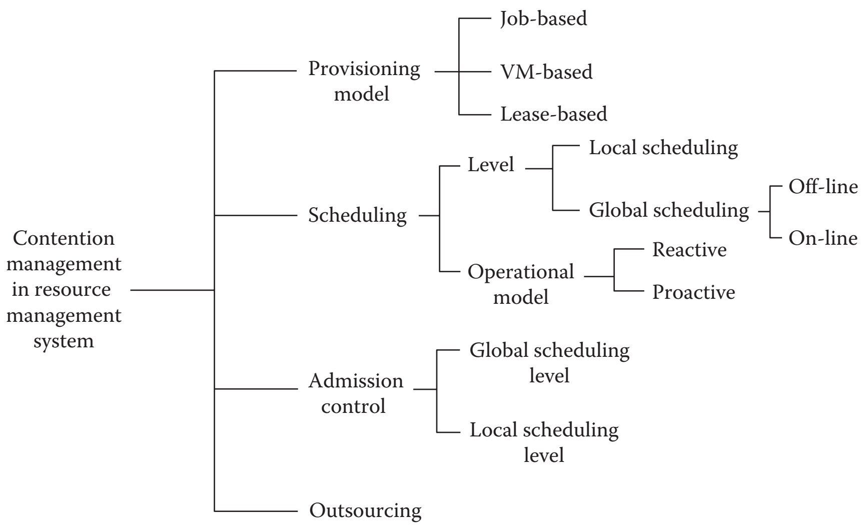

FIGURE 57.4 Components of a resource management system and their approach for dealing with contentions.

the job. Job model resource provisioning has been widely employed in distributed systems. However, this model cannot perfectly support all resource contention solutions.

Job-based systems provision hardware for jobs while they offer a limited support for software availability. In fact, in job-based model users do not have root access, therefore it is difficult to install and use required software packages. Many job-based systems support availability based on queuing theory along with scheduling algorithms. However, queue-based systems usually do not assure specific time availabilities.

To support availability and hardware dimensions, Nurmi et al. [68], present advance reservation (AR) model over the job-based provisioning model. They support AR through predicting waiting time of jobs in the queue. Hovestadt et al. [45] propose plan-based scheduling (opposite to queue-based) that finds the place of each job (instead of waiting in the queue) to be able to support AR model. In this system, on the arrival of each job the whole schedule is re-planned to optimize the resource utilization.

Falkon [75], Condor glidin [34], MyCluster [108], and Virtual Workspace [53] have applied a multilevel/hierarchical scheduling on top of a job-based system to offer other provisioning models (such as lease-based model, which is described in Section 57.4.1.3). In these systems, one scheduler allocates resources to another scheduler and the other scheduler runs the jobs on the allocated resources.

57.4.1.2 Virtual Machine Model

Virtual machines (VMs) are considered as an ideal vehicle for resource provisioning in distributed systems. The reason is that, in VM model, unlike the job model, hardware, software, and availability can be provisioned for user requests. Additionally, VMs' capability in getting suspended, resumed, or migrated without major utilization loss have proved to be useful in resource management. Therefore, VM-based provisioning model is commonly used in current distributed systems.

The VM-based resource provisioning model is used in creating virtual Clusters on top of an existing infrastructure. Virtual clusters (VC) are usually utilized for job-based batch processing. For example, in MOSIX [13], Clusters of VMs are transparently created to run high performance computing (HPC) applications. The Nimbus toolkit [52] provides "one-click virtual Cluster" automatically on heterogeneous sites through contextualizing disk images. Amazon EC2, provides VM-based Cluster instances* that offer supercomputing services to expedite execution of HPC applications, without delaying the user in a queue or acquire expensive hardware. Automatic VM creation and configuration in short time is also considered in In-VIGO [2] and VMplants [56]. An extension of Moab [29] creates VM-based virtual Clusters to run HPC batch applications.

Many commercial datacenters use VM-based provisioning model to provide their services to resource consumers. Such datacenters offer services such as Virtual Cluster, or hosting servers including web, email, and DNS.

Datacenters usually contain large scale computing and storage resources (order of 100s–1000s) and consume so much energy. A remarkable benefit of deploying VM-based provisioning model in datacenters is the consolidation feature of VMs that can potentially saves the energy consumption [105]. However, VM consolidation requires accurate workload prediction in the datacenters. Moreover, the consolidation impact on service level agreements (SLA) needs to be considered. VM consolidation can be performed in a static (also termed cold consolidation) or dynamic (hot consolidation) manner. In the former, VMs needs to be suspended and resumed on another resource that involves time overhead. In the latter approach, live migration [107] of VMs is used, thus, is transparent from the user.

Solutions such as VMware, Orchestrator, Enomalism, and OpenNebula [32] provide resource management for VM-based data centers.

There are also concerns in deploying VM-based provisioning model and Virtual Clusters. Networking and load balancing among physical Clusters is one of the challenges that is considered in Vio-Cluster [79]. Power efficiency aspect and effectively utilizing VMs capability in suspending and migrating are also

* http://aws.amazon.com/hpc-applications/

considered by many researchers [51,66,106]. Overhead and performance issues involved in applying VMs to run compute-intensive and IO-intensive jobs, fault tolerance, and security aspects of VMs are also of special importance in deploying VM-based provisioning model.

57.4.1.3 Lease Model

This model is considered as an abstraction for utility computing in which the user is granted a set of resources for specific interval and agreed quality of service [39]. In this model, job execution is independent from resource allocation, whereas in the job model resource allocation is the consequence of running a job.

Formally, a *lease* is defined by Sotomayor [93] as "a negotiated and renegotiable contract between a resource provider and a resource consumer, where the former agrees to make a set of resources available to the latter, based on a set of lease terms presented by the resource consumer." If lease extension is supported by resource management system, then users would be able to extend their lease for a longer time. This is particularly useful in circumstances that users have inaccurate estimation of required time. Virtual machines are suitable vehicles to implement lease-based model. Depending on the contract, resource procurement for leases can be achieved from a single provider or from multiple providers.

57.4.2 Scheduling Unit

The way user requests are scheduled in an interconnected distributed system affects types of resource contentions occurring. Efficient scheduling decisions can prevent resource contention or reduce its impact whereas poor scheduling decisions can lead to more resource contentions.

In an interconnected distributed system, we can recognize two levels of scheduling, namely, local (domain level) scheduling and global scheduling (meta-scheduling). The global scheduler is generally in charge of assigning incoming requests to resource providers within its domain (e.g., Clusters or sites). In the next step, the local scheduler performs further tuning to run the assigned requests efficiently on resources.

From the resource contention perspective, scheduling methods can either react to resource contention or proactively prevent the resource contention to occur.

57.4.2.1 Local Scheduling

Local scheduler deals with scheduling requests within each distributed system (e.g., Cluster or site). Scheduling policies in this level can mainly deal with request-initiated and origin-initiated contentions. Indeed, there are few local schedulers that handle interdomain-initiated contention.

Backfilling is a common scheduling policy in local resource management systems (LRMS). The aims of backfilling are increasing resource utilization, minimizing average request response time, and reducing queuing fragmentation. In fact, backfilling is an improved version of FCFS in which requests that arrive later, possibly are allocated earlier in the queue, if there is enough space for them. Variations of backfilling policy are applied in local schedulers:

- Conservative: In which a request can be brought forward if it does not delay any other request in the queue.
- Aggressive (EASY): The reservation of the first element in the queue cannot be postponed. However, the arriving request can shift the rest of scheduled requests.
- Selective: If the slowdown of a scheduled request exceeds a threshold, then it is given a reservation, which cannot be altered by other arriving requests.

There are also variations of backfilling method that are specifically designed to resolve request-initiated resource contentions. Snell et al. [91] applied preemption on backfilling policy. 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.

Multiple resource partitioning is another scheduling approach for local schedulers by Lawson and Smirni [59]. In this approach, resources are divided into partitions that potentially can borrow resources from each other. Each partition has its own scheduling scheme. For example, if each partition uses EASY backfilling, then one request from another QoS level can borrow resources, if it does not delay the pivot request of that partition.

In FCFS or backfilling scheduling policies, the start time of a request is not predictable (not determined). Nonetheless, in practice, we need to guarantee timely access to resources for some requests (e.g., deadline-constraint requests in a QoS-based system). Therefore, many local schedulers support Advance Reservation (AR) allocation model that guarantees resource availability for a requested time period. Resource management systems such as LSF, PBSPRO, and MAUI support AR.

Advance Reservation is prone to low-resource utilization specially if the reserved resources are not used by the users. Additionally, it increases the response time of normal requests [63,90]. These side-effects of AR can be minimized by limiting the number of AR, and leveraging flexible AR (in terms of start time, duration, or number of processing elements needed).

57.4.2.2 Global Scheduling (Meta-Scheduling)

Global scheduler in an interconnected distributed system usually has two aspects. On the one hand, the scheduler is in charge of assigning incoming requests to resource providers within its domain (e.g., Clusters). On the other hand, it is responsible to deal with other distributed systems such as schedulers or gateways that delegate other peer distributed systems. This aspect of global schedulers can particularly resolve interdomain-initiated and origin-initiated resource contentions.

The global scheduler either works off-line (i.e., batches incoming requests and assigns each batch to a Cluster), or is online (i.e., assign each request to a local scheduler as it is received). The global schedulers can proactively prevent resource contentions.

57.4.3 Admission Control Unit

Controlling the admission of requests prevents the imbalanced deployment of resources. By employing an appropriate admission control policy different types of resource contentions can be avoided. An example of the situation without admission control in place is when two requests share a resource but one of them demands more time. In this situation, the other request will face low-resource availability and subsequently, high response time. Thus, lack of admission control can potentially lead to request-initiated contention.

Admission control behavior should depend on the workload condition in a resource provider. Applying a strict admission control in a lightly loaded system results in low resource utilization and high rejection of requests. Nonetheless, the consequence of applying less strict admission control in a heavily loaded resource is more QoS violation and less user satisfaction [112].

Admission control can function in different ways. To tackle request-initiated contention, admission control commonly carried out via introducing a valuation function. The valuation function relates the quality constrains of users to a single quantitative value. The value indicates the amount a user is willing to pay for a given quality of service (QoS). Resource management system use the valuation functions to allocate resources with the aim of maximizing aggregate valuation of all users.

Admission control also can be applied in interdomain-initiated contentions to limit the amount of admitted requests of each organization to be proportional to their resource contribution. Similarly, admission control can be applied to avoid origin-initiated resource contention. For this purpose, admission control policy would not admit external requests where there is peak load of local requests.

Placement of admission control component in a resource management system of a interconnected distributed system can be behind the local scheduler and/or behind the global scheduler. In the former, for rejecting a request there should be an alternative policy to manage the rejected request. In fact, rejecting by a local scheduler implies that the request has already been admitted and, hence, has to be taken care. For instance, the rejected request can be redirected to another resource provider or even queued in a separate queue to be scheduled later. Deploying admission control behind the global scheduler is easier in terms of managing the rejected requests. However, the drawback of employing admission control with global scheduler is that the global scheduler may not have updated information about site's workload situation.

57.4.4 Outsourcing Unit

Interconnectivity of distributed systems creates the opportunity to resolve the resource contention via employing resources from other distributed systems. Therefore, resource management systems in interconnected distributed computing systems usually have a unit that decides about details of outsourcing requests (i.e., redirecting arriving requests to other distributed systems) such as when to outsource and which requests should be outsourced. In terms of implementation, in many systems, the outsourcing unit is incorporated into either admission control or scheduling unit. However, it is also possible to have it as an independent unit in the resource management system.

Outsourcing is generally applied when there is a peak demand or there is a resource contention (specially request-initiated contention). In this situation to serve requests without resource contention, some requests (e.g., starved requests) are selected to be redirected to other distributed systems.

Cloud computing providers have been of special interest to be employed for outsourcing (off-loading) requests [84]. This issue has pushed the emergence of hybrid clouds, which are a combination of a private (organizational) Cloud and public Clouds.

57.4.5 Preemption Mechanism

Preemption mechanism in a resource management system makes the resources free and available for another, possibly higher priority, request. Preemption is a useful mechanism to resolve request-initiated and origin-initiated contentions. Preemption of a running process can be performed manually or automatically through the resource management system.

The way preemption mechanism is implemented, depends on the way checkpointing operation is carried out. If the checkpointing is not supported, then the preempted process has to be killed and restarted at a later time. If checkpointing is supported (both by the running process and by the scheduler), then the preempted request can be suspended and resumed at a later time. However, checkpointing is not a trivial task in distributed systems. We will deal with checkpointing hurdles in Section 57.4.5.4.

Due to the critical role of preemption in solving different types resource contentions, in this section, we investigate preemption in distributed systems from different angles. Particularly, we consider various usages of preemption and the way they solve resource contentions. Then, we investigate possible side-effects of preemption. Finally, we discuss how a preempted request (i.e., job/VM/lease) can be resumed in a distributed system.

57.4.5.1 Applications of Preemption Mechanism

Preemption in distributed systems can be applied for reasons that are presented in Figure 57.5. As we can see, preemptions can be used to resolve resource contention. However, there are other usages of preemption in distributed systems that we will discuss them in this part.

Preemption is used to resolve request-initiated resource contentions. One approach is employing preemption in local scheduler along with the scheduling policy (e.g., backfilling) to prevent unfairness. For instance, when a backfilled request exceeds the allocated time slot and interferes with the reservation of other requests preemption mechanism can preempt the backfilled requests and therefore the reservations can be served on time. The preempted request can be allocated another time slot to finish its computation [40].

A preemptive scheduling algorithm is implemented in MOSIX [4] to allocate excess (unclaimed) resources to users that require more resources than their share. However, these resources will be released

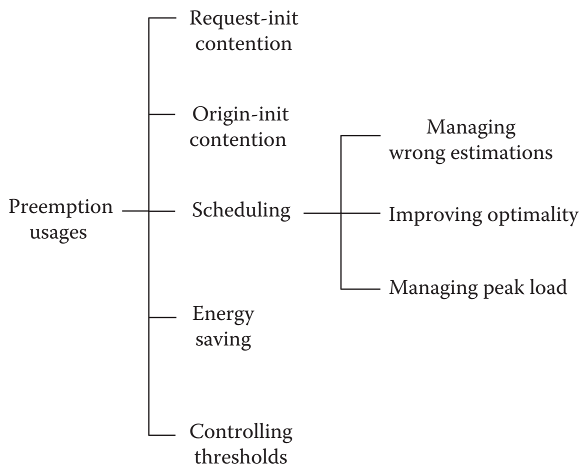

FIGURE 57.5 Different usages of preemption in distributed systems.

as soon as they are reclaimed. MOSIX also support situation that there are local and guest jobs and can consider priority between them (origin-initiated contention).

Scojo-PECT [92] provides a limited response time for several job classes within a virtualized Cluster. It employs DiffServ solution that is implemented via coarse-grained preemption to cope with the request-initiated resource contention. The preemptive scheduler aims at creating a fair-share scheduling between different job classes of a Grid. The scheduler works based on a coarse-grained time sharing and for preemption it suspends VMs on the disk.

Walters et al. [109] introduced a preemption-based scheduling policy for batch and interactive jobs within a virtualized Cluster. In this work, batch jobs are preempted in favor of interactive jobs. The authors introduce different challenges in preempting jobs including selecting a proper job to be preempted, checkpointing the preempted job, VM provisioning, and resuming the preempted job. Their preemption policy is based on weighted summation of factors such as the time spent in the queue.

Haizea [93] is a lease scheduler that schedules a combination of advanced reservation and best effort leases. Haizea preempts best effort leases in favor of advance reservation requests. Haizea also considers the overhead time imposed by preempting a lease (suspending and resuming included VMs).

Preemption of parallel jobs has also been implemented in the Catalina job scheduler [63] in San-Diego Supercomputer Center (SDSC). They have added preemption to conservative backfilling. The job preemption is carried out based on job priorities which is determined based on weighted summation of factors such as the time a request waits in the queue, the size (number of processing elements) required by the request, and expansion factor of the request. In general, the policy tries to preempt jobs that require fewer processing elements because they impose less overhead to the system for preemption. In fact, preempting jobs with larger size (wide jobs) implies more overhead because of the time needed for saving messages between nodes.

Isard et al. [49] have investigated the problem of optimal scheduling for data intensive applications, such as Map-Reduce, on the Clusters where the computing and storage resources are close together. To achieve the optimal resource allocation, their scheduling policy preempts the currently running job in order to maintain data locality for a new job.

Preemption can be applied to resolve the origin-initiated resource contentions. Ren et al. [76] have proposed a prediction method for unavailable periods in fine-grained cycle sharing systems where there are mixture of local jobs and global (guest) jobs. The prediction is used to allocate global requests in a way that do not disturb local requests.

Gong et al. [38] have considered preemption of external tasks in favor of local tasks in a Network of Workstations (NOW) where local tasks have preemptive priority over external tasks. They provided a performance model to work out the run time of an external task that is getting preempted by local tasks in a single processor. The performance model also covers the average runtime of the whole external job which is distributed over NOW.

There are other research works such as [5,6,81,82] that apply preemption for removing origin-initiated contentions.

Apart from removing resource contention, preemption has other usages in resource management systems. More importantly, preemption can be applied to improve the quality of scheduling policies. In fact, preemption can be used as a tool by scheduler to enforce its policy.

Scheduling algorithms in distributed systems are highly dependent on user runtime estimation. There are studies (e.g., [99]) that demonstrate the inefficiency of these estimations and how these wrong estimation can compromise the scheduling performance. In the presence of inaccurate estimations, preemption can be deployed to help the scheduler in enforcing its decision through preempting the process that has wrong estimations. Particularly, this is critical for systems that support strict reservation model such as advanced reservation. In this situation, preemption abstracts the scheduling policy from the obstacles in enforcing that policy [44].

Preemption can be applied to improve the optimality of resource scheduling. Specifically, online scheduling policies are usually not optimal because jobs are constantly arriving over time and the scheduler does not have a perfect knowledge about them [4]. Therefore, preemption can potentially mitigate the nonoptimality of the scheduling policy.

Preemption mechanism can be employed for managing peak load. In these systems, resource-intensive applications or batch applications are preempted to free the resources during the peak time. Accordingly, when the system is not busy and the load is low, the preempted requests can be resumed [69].

Preemption can be employed to improve the system and/or user centric criteria, such as resource utilization and average response time. Kettimuthu et al. [54] have focused on the impact of preempting parallel jobs in supercomputers for improving the average and worst-case slowdown of jobs. They suggest a preemption policy, called *Selective Suspension*, where an idle job can preempt a running job if the suspension factor is adequately more than the running job.

A recent application of preemption is in energy conservation in datacenters. In fact, one prominent approach in energy conservation of virtualized datacenters is VM consolidation, which takes place when resources in the datacenter are not utilized efficiently. In VM consolidation, VMs running on under-utilized resources are preempted (suspended) and resumed on other resources. VM consolidation can also occur through live migration of VMs [107] to minimize the unavailability time of the VMs. When a resource is evacuated, it can be powered off to reduce the energy consumption of the datacenter.

Salehi et al. [83] have applied VM preemption to save energy in a datacenter that supports requests with different SLAs and priorities. They introduce an energy management component for Haizea [93] that determines how resources should be allocated for a high-priority request. The allocation can be carried out through preempting lower-priority requests or reactivating powered off resources. The energy management component can also decide about VM consolidation, in circumstances that powered on resources are not being utilized efficiently.

Preemption can be used for controlling administrative (predetermined) thresholds. The thresholds can be configured on any of the available metrics. For instance, the temperature threshold for CPUs can be established that leads to the system automatically preempts part of the load and reschedule on other available nodes. Bright Cluster Manager [1] is a commercial Cluster resource management system that offers the ability to establish preemption rules by defining metrics and thresholds.

57.4.5.2 Preemption Challenges

Operating systems of single processor computers have been applying preemption mechanism for a long time to offer interactivity to the end-user. However, since interactive requests are not prevalent in distributed systems, there has been less demand for preemption in these systems. More importantly, achieving preemption in distributed systems entails challenges that discourage researchers to investigate deeply on that. This challenges are different based on the resource provisioning model.

TABLE 57.1 Preemption Challenges in Different Resource Provisioning Models

	Resource Provisioning Model		
Challenge	Job-Based	VM-Based	Lease-Based
Coordination	✓	✓	✓
Security	✓	×	×
Checkpointing	✓	×	×
Time overhead	✓	✓	✓
Permission	✓	✓	×
Impact on queue	✓	✓	✓
Starvation	✓	✓	✓
Preemption candidates	✓	✓	✓

In this part, we present the detailed list of challenges that distributed systems encounter in preempting requests in various resource provisioning models. Moreover, a summary of preemption challenges based on different provisioning models is provided in Table 57.1.

- *Coordination*: Distributed requests (jobs/VMs/leases) are scattered on several nodes by nature. Preemption of the distributed requests have to be coordinated between the nodes that are executing them. Lack of such coordination leads to inconsistent situation (e.g., because of message loss) for the running request.
- *Security*: Preemption in job-based systems implies security concerns regarding files that remain open and swapping-in the memory contents before job resumption. In other words, in job-based systems operating system has to provide the security of not accessing files and data of the preempted processes. Since VM- and lease-based systems are self-contained (isolated) by nature, there is not usually security concern in their preemption.
- *Checkpointing*: Lack of checkpointing facilities is a substantial challenge in job-based resource provisioning model. Because of this problem, in job-based systems the preempted job is generally killed, which is a waste of resources [91]. Checkpointing problem is obviated in VM and lease-based resource provisioning models [94]. Due to the fundamental role of checkpointing for preemption mechanism, in Section 57.4.5.4 we discuss it in details.
- *Time overhead*: In VM- and lease-based resource provisioning models, time overhead imposed to the system to perform preemption is a major challenge. If preemption takes place frequently and the time overhead would not be negligible, then the resource utilization will be affected.

  Additionally, disregarding the preemption time overhead in scheduling, prevents requests to start at the scheduled time [94]. In practice, resource management systems that support preemption, must have an accurate estimation of the preemption time overhead. Overestimating the preemption time overhead results in idling resources. However, underestimating the preemption time overhead ends up in starting leases with delay, which subsequently might violate SLA agreements.

  Sotomayor et al. [94] have presented a model to predict the preemption time overhead for VMs. They identified that the size of memory that should be de-allocated, number of VMs mapped to each physical node, local or global memory used for allocating VMs, and the delay related to commands being enacted are effective on the time overhead of preempting VMs. To decrease the preemption overhead, the number of preemptions that take place in the system has to be reduced [87].
- *Permission:* In the lease-based resource provisioning model, preempting leases is not allowed by default. In fact, one difference between lease-based and other resource provisioning models is that jobs and VMs can be preempted without notifying the user (requester), whereas leases require the requester's permission for preemption [39]. Therefore, there must be regulations in the lease

terms to make lease preemption possible. These terms can be in the form of QoS constraints of the requests or can be bound to pricing schemes. For instance, requests with tight deadline, advance reservations, or requests with tight security possibly choose to pay more instead of getting preempted while they are running.

- *Impact on other requests*: Most of the current distributed systems use a variation of backfilling policy as the scheduling policy. In backfilling, future resource availabilities are reserved for other requests that are waiting in the queue. Preempting the running process and allocating resources to a new request affects the running job/lease as well as the reservations waiting in the queue. Re-scheduling of the preempted requests in addition to the affected reservations are side-effects of preemption in distributed systems.
- *Starvation*: Preemption leads to increasing the response time and, in the worst case, starvation for low-priority requests [5]. There is a possibility that low-priority requests get preempted as soon as they start running. This leads to unpredictable waiting time and unstable situation for low-priority requests. Efficient scheduling policies can prevent unstable and long waiting time situation. One approach to cope with the starvation challenge is restricting the number of requests admitted in a distributed system. Salehi et al. [82] have proposed a probabilistic admission control policy that restricts the queue length for low-priority requests in a way that they would not suffer from starvation.
- *Preemption candidates*: By allowing preemption in a distributed system, there is a possibility that several low priority requests have to be preempted to make sufficient vacant resources for the high-priority request. Therefore, there are several sets of candidate requests whose preemption can create adequate space for the high-priority request. As it is expressed in Figure 57.6, there are several candidate sets (Figure 57.6b) that their preemption can vacate resources for the required time interval (i.e., from t_1 to t_2 as indicated in Figure 57.6a).

  Selecting distinct candidate sets affects the amount of unused space (also termed scheduling fragment) appear in the schedule. Furthermore, preempting different candidate sets imposes different time overhead to the system because of the nature of the requests preempted (e.g., being data-intensive). In this situation, choosing the optimal set of requests for preemption is a challenge that needs to be addressed.

  To cope with this challenge, backfilling policy has been extended with preemption ability in Maui scheduler [91] to utilize scheduling fragments. Salehi et al. [5] have proposed a preemption policy that determines the best set of leases to be preempted with the objective of minimizing preemption

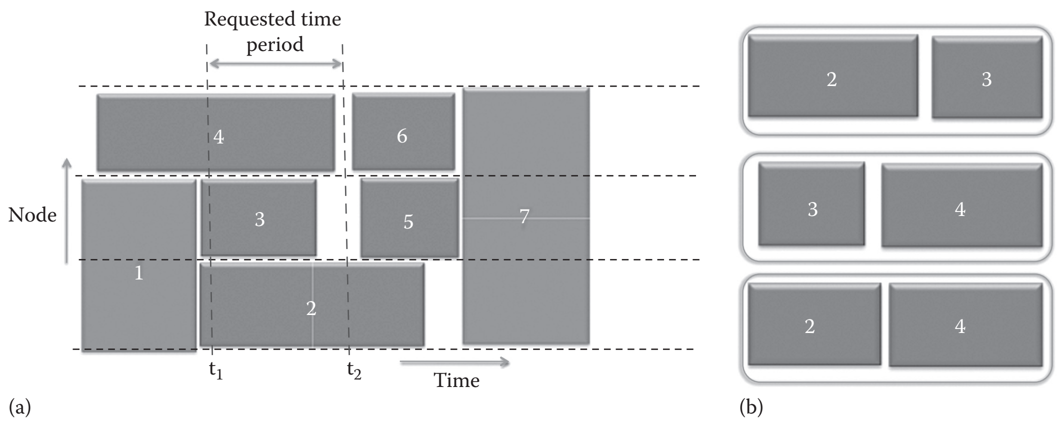

FIGURE 57.6 Preemption candidates for a request that needs two nodes. (a) Shows collision of the requested time interval with running requests within a scheduling queue. (b) Presents different candidate sets that their preemption creates space for the new request.

AQ1

time overhead. A preemption policy is also presented by Walter et al. [109] in a VM-based system with the objective of avoiding starvation for batch requests where a combination of batch and interactive requests co-exist in the system.

57.4.5.3 Possibilities for Preempted Requests

Issues discussed thus far are related to preemption mechanism and its challenges. However, making a proper decision for the preempted request is also important. This decision depends on the facilities provided by the resource management system of a distributed system. For example, migration is one choice that is viable in some distributed systems but not in all of them.

Thanks to the flexibility offered by deploying VM-based resource provisioning models, resource managers are capable of considering various possibilities for the preempted request. Nonetheless, in job-based systems, if preemption is possible, the possible action on the preempted job is usually limited to killing or suspending and resuming of the preempted job. Over the next paragraphs, we introduce various cases that can possibly happen for preempted VMs/leases. Additionally, in Figure 57.7 it is expressed that how different possibilities for the preempted VM affect the VMs' life cycle.

- *Cancelling*: VMs can be canceled (terminated) with/without notifying the request owner. VMs offered in this fashion are suitable for situation that the resource provider does not have to guarantee the availability of the resources for a specific duration. Spot instances offered by Amazon EC2 is an example of cancelling VMs. Isard et al. [49] have used cancelling VMs to execute map-reduce requests. Cancelling VMs imposes the minimum overhead time that is related to the time needed to terminate VMs allocated to the request.

  In job-based systems, cancelling (killing) jobs is a common practice [91] because of the difficulty of performing other possible actions.
- *Restarting*: In both job-based and VM-based systems, the preempted request can be killed (similar to cancelling) and restarted either on the same resource or on another resource. The disadvantage of this choice is losing the preliminary results and wasting the computational power. Restarting can be applied for best-effort and deadline-constraint requests. In the former, restarting can be performed at any time whereas, in the latter, deadline of the request has to be taken into account for restarting.

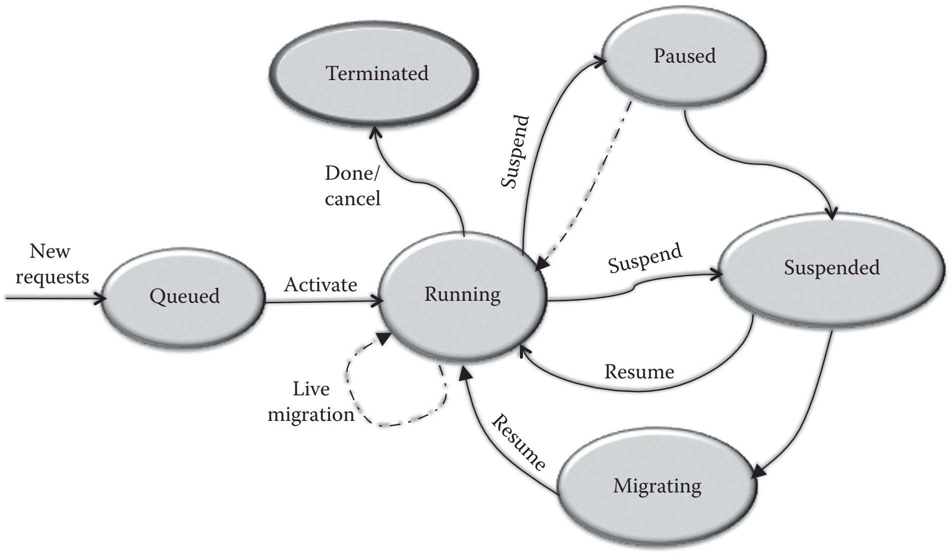

FIGURE 57.7 VM life cycle by considering different possible preemption decisions in a resource management system.

- *Malleability* (*partial preemption*): In this manner, the number of nodes/VMs allocated to a request might be changed while it is executing. In this approach, the request should be designed to adapt dynamically to the changes. This action can be applied on malleable jobs [72] in job-based systems. In VM and lease-based systems, frameworks such as Cluster-on-Demand (COD) [64], support this manner of preemption via cancelling some of the VMs of a lease. Malleability is also known as partial-preemption and can be used to implement dynamic partitioning (see Section 57.3.1).
- *Pausing*: When a VM is paused, it does not get any CPU share, however, it remains in the memory. Resumption of the VM, in this case, happens by getting CPU share and, thus, is very fast (Figure 57.7). Hence, we cannot consider pausing as a complete preemption action.

  Nonetheless, the main usage of pausing is to perform lease-level preemption. In preempting a lease (several correlated VMs), to prevent inconsistency or message loss, first, all VMs are paused and then, suspension takes place [44] (link between pause state and sleep (suspended) state in Figure 57.7). In Section 57.4.5.4, we discuss how pausing VMs helps in preempting leases.
- *Suspending*: When a VM is suspended, the entire state of the VM (including the state of all processes running within the VM) is saved to the disk. At resumption time, the VM continues operating from the suspended point. The suspended request has to be rescheduled to find another free time slot for the remainder of its execution. In job-based systems, the operating system should retain the state of the preempted process and resume the job.

  An important question after suspension is where to resume a VM/lease? Answering this question is crucial particularly for data-intensive applications. A suspended request can be resumed in one of the three following ways:
  1. Resuming on the same resource; this case does not yield to optimal utilization of whole resources.
  2. Resuming on the same site but not essentially on the same resource; In this case, usually data transfer is not required.
  3. Resuming on different site: This case leads to migrating to another site, which entails data transfer. This is, particularly, not recommended for data-intensive requests.
- *Migrating:* VMs of the preempted request are moved to another resource provider to resume the computation (also called cold migration). According to Figure 57.7, migrating involves suspending, transferring, and resuming VMs. Transferring overhead in the worst case includes transferring the latest VM state in addition to the disk image of the VM. One solution to mitigate this overhead is migrating to another site within the same provider which has a high bandwidth connection available (e.g., within different Clusters of a datacenter). In terms of scheduling, multiple reservation strategies can be applied to assure that the request will access resources in the destination resource provider [93].
- *Live Migration:* Using live migration preemption can be carried out without major interruption in running VMs involved in preemption (see live migration link in Figure 57.7). This is particularly essential in conditions that no interruption can be tolerated (e.g., Internet servers). For this purpose, the memory image of the VM is transferred over the network. There are techniques to decrease the live migration overhead, such as transferring just the dirty pages of the memory.

Apart from the aforementioned choices, there are requests that cannot be preempted (i.e., nonpreemptable requests). For example, critical tasks in workflows that have to start and finish at exact times to prevent delaying the execution of the workflow [57]. Another example is secure applications that cannot be moved to any other provider and cannot also be interrupted in the middle of execution.

In a particular resource management system, one or combination of the mentioned actions can be performed on the preempted request. The performed action can be based on the QoS constraints of the requests or restrictions that user declares in the request. Another possibility is that the resource management system dynamically decide the appropriate action on the preempted request.

57.4.5.4 Checkpointing in Distributed Systems

Checkpointing is the function of storing the latest state of a running process (e.g., job, VM, and lease). Checkpointing is an indispensable part of preemption, if the preempted request is going to resume its execution from the preempted point. In fact, checkpointing is the vehicle of implementing preemption. Apart from preemption, checkpointing has other usages including providing fault-tolerance for the requests.

Checkpointed process can be stored on a local storage, or carried over the network to a backup machine for future recovery/resume. Checkpointing has to be achieved in an *Atomic* way, which means either all or none of the modifications are checkpointed (transferred to the backup machine). There are various approaches to achieve checkpointing which are presented briefly in Figure 57.8. In this section, we explain checkpointing strategies for different provisioning models in distributed systems.

57.4.5.4.1 Checkpointing in Job-Based Provisioning Model

Checkpointing approaches are categorized as application-transparent and application-assisted (see Figure 57.8). In application-assisted (user-level) checkpointing, the application defines the necessary information (also called critical data area) that have to be checkpointed. The disadvantage of this approach is that it entails modifying the application by the programmer. However, this approach imposes little overhead to the system because it just checkpoints the necessary parts of the application; additionally, the frequency of performing checkpointing is determined by the user. User-level checkpointing can be further categorized as follows:

- Source-code level: In this manner, checkpointing codes are hard-coded by developers. However, there are some source code analysis tools [21,30] that can help developers to figure out the suitable places that checkpointing codes can be inserted.
- Library level: There are ready-made libraries for checkpointing, such as Libckpt [74] and Condor libraries [62]. To use this kind of checkpointing, developers have to recompile the source code by including the checkpointing library in their program.

As noted in Figure 57.8, checkpointing can also be done in application-transparent manner. This approach is also known as system level, Operating System level, or kernel level in the literature. As the name implies, in this approach the application is not aware of checkpointing process. Therefore, the application does not need to be modified to be checkpointable. Application-transparent checkpointing technique is particularly applied in preemption whereas application-assisted scheme is more used in fault-tolerance techniques. Examples of system level checkpointing in the system level are BLCR [41] and CRAK [111].

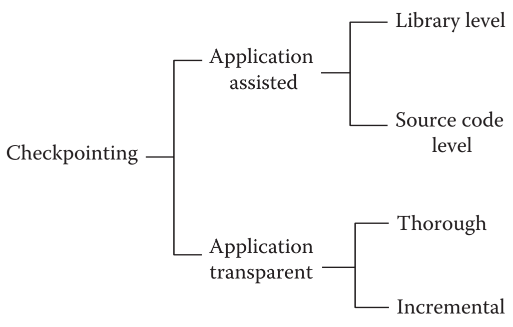

FIGURE 57.8 Checkpointing methods in distributed systems.

Since the application-transparent checkpointing methods have to checkpoint the whole application state, they impose significant time overhead to the system. Another drawback of this approach is that the system-level checkpointing methods are dependent on a specific version of the operating system that they are operating on and, hence, are not entirely portable.

In order to mitigate the checkpointing overhead, incremental checkpointing technique is used [43] in which just the changes since the previous state are checkpointed. Typically, a page-fault technique is used to find the dirty pages and write them to the backup [43,50].

Checkpointing of the distributed applications that run in a distributed system, such as a Cluster, is more complicated. For these applications, not only the state of the application on each running node should be checkpointed, but it has to be assured that the state of the whole application across several nodes remains consistent. Therefore, the checkpointing process across nodes that run the application must be synchronized in a way that there would be neither message loss nor message reordering. Checkpointing of the distributed applications (also termed coordinated checkpointing) traditionally is developed based on the global distributed snapshot concept [22]. These solutions are generally application-level, dependent on a specific version of operating system, and also dependent on the platform implementation (e.g., MPI implementation). Cocheck [95], BLCR [41], and MPICHV [24] are examples of these solutions.

There are various approaches for managing the connections between processes running on different nodes while the checkpointing is performed. In MPICHV [24], the connection among processes has to be disconnected before each process saves its local state to the checkpoint file. In this approach, connections should be re-established before processes can resume their computation. Another approach, which is used in LAM/MPI, uses bookmarking mechanism between sender and receiver processes to guarantee message delivery at the checkpointing time.

57.4.5.4.2 Checkpointing in VM-Based Systems

Virtualization technique provides application-transparent checkpointing as an inherent feature that involves saving (suspending) and restoring (resuming) of the VM state [14,20,55].

In a virtualized platform, hypervisor (also called virtual machine monitor) is an essential component that manages different VMs concurrently running on the same host. Generally, the hypervisor is in charge of VM checkpointing. To checkpoint a VM, its internal state including memory, cache, and data related to the virtual devices have to be stored on the disk. Disk image snapshot also has to be stored, specially when the checkpointed VM is transferred and sharing image is not possible. Current virtual machine monitors, such as VMware, Xen, and KVM, support saving/restoring the state of VMs to/from a file. However, taking a copy of the disk image is not practically possible because of the size of the disk [69]. Therefore, currently, checkpointing is mostly carried out within resources with a shared storage, such as NFS.

Accordingly, distributed applications running on VMs across several nodes within a Cluster can be checkpointed [46]. Checkpointing of such applications is complicated because of the possible correlations between VMs (e.g., TCP packets and messages exchanged between VMs). The checkpointing process should be aware of these correlations, otherwise the checkpointing process leads to inconsistency in running the distributed applications.

To handle the checkpointing, when a checkpointing event is initiated, all the nodes that run a process of the distributed application receive the event. Upon receiving the event, the hypervisor *pauses* computation within VMs in order to preserve the internal state of VM and also to stop submitting any new network message (see Figure 57.7). In the next step, checkpointing protocols save the in-flight messages (i.e., network packets). For this purpose, the hypervisor collects all the incoming packets and queue them. Finally, a local VM checkpointing is performed through which the VM's internal state, VM disk image, and queued messages for that VM are saved in the checkpoint file [44].

57.5 Contention Management in Practice

Various types of distributed systems for resource sharing and aggregation have been developed. They include Clusters, Grids, and Clouds. In this section, we study these systems from the resource contention perspective. We identify and categorize properties of the reviewed systems and summarize them in Table 57.2 for Clusters and in Table 57.3 for Grids and Clouds.

57.5.1 Contention Management in Clusters

Compute Clusters are broadly categorized as dedicated and shared Clusters. In dedicated Clusters a single application exclusively runs on the Cluster's nodes. Mail servers, and web servers are examples of dedicated Clusters.

By contrast, in a shared Cluster the number of requests is significantly higher than the number of Cluster nodes. Therefore, nodes have to be shared between the requests by means of a resource management system [100]. From the resource contention perspective, shared Clusters are generally prone to request-initiated contention.

Virtual Clusters are another variation of Clusters that work based on VMs. Although users of these Clusters are given root access to the VMs, these resources are not dedicated to one user in hardware level (i.e., several VMs on the same node can be allocated to different users).

A multicluster is an interconnected distributed system that consists of several Clusters possibly in different organizations. Multiclusters are prone to origin-initiated contentions as well as request-initiated contention.

TABLE 57.2 Role of Different Components of Cluster Management Systems in Dealing with Resource Contentions

System	Provisioning Model	Operational Model	Context	Contention Initiation	Contention Management	RMS Component
Haizea [93]	Lease	Reactive	Cluster	Request	Preemption	Local sched
VioCluster [79]	VM	Proactive and reactive	MultiCluster	Origin	Preemption	Local sched
Snell et al. [91]	Job	Reactive	Cluster	Request	Preemption	Local sched
Lawson Smirni [59]	Job	Proactive and reactive	Cluster	Request	Partitioning	Local sched
Walters et al. [109]	VM	Reactive	Cluster	Request	Preemption	Local sched
Scojo-PECT [92]	VM	Reactive	Cluster	Request	DiffServ and preemption	Local sched
MOSIX [4]	VM	Reactive	Cluster	Origin	Fairness and (preemption)	Local sched
Sharc [100]	Job	Reactive	Cluster	Request	Partitioning	Local sched and Admission ctrl
COD [64]	Lease	Reactive	Cluster	Request	Partial preemption	Local sched
Cluster reserves [10]	Job	Proactive	Cluster	Request	Partitioning	Local sched
Muse [23]	Job	Reactive	Cluster	Request	Economic (utility)	Local sched
Shirako [48]	Lease	Reactive	MultiCluster	Interdomain	Token	Local sched
Lee et al. [60]	Job	Proactive and reactive	MultiCluster	Request	Fairness	Global and Local sched
MUSCLE [42]	Job	Proactive	MultiCluster	Request	Utility	Global sched
Percival et al. [73]	Job	Reactive	Cluster	Request	Economic (utility)	Admission ctrl

AQ2

AQ3

TABLE 57.3 Role of Different Components of Grid/Cloud Resource Management Systems in Dealing with Resource Contentions

System	Provisioning Model	Operational Model	Context	Contention Initiation	Contention Management	RMS Component
GridWay [103]	Job	Reactive	Grid Federation	Request	Outsourcing	Global sched (outsourcing)
Amazon Spot Market	VM	Reactive	Cloud	Request	Economic (auction)	Global sched
Van et al. [101]	VM	Reactive	Cloud	Request	Economic (utility)	Global sched
OurGrid [9]	Job	Reactive	Grid	Interdomain	Incentive	Global sched
Ren et al. [76]	Job	Proactive	Desktop Grid	Origin	Preemption	Local sched
Salehi et al. [6]	Lease	Proactive	Grid Federation	Origin	Preemption	Global sched
InterGrid [26]	Lease	Proactive	Grid Federation	Origin	Partitioning	Global sched
InterCloud [97]	VM	Reactive	Cloud Federation	Request	Economic (utility)	Admission ctrl
RESERVOIR [78]	VM	Reactive	Cloud Federation	Request	Outsourcing	Outsourcing
Sandholm et al. [86]	VM	Reactive	Grid	Request	Partial preemption	Admission ctrl
InterGrid Peering [26]	Lease	Reactive	Grid Federation	Interdomain	Global scheduling	Global sched
Salehi et al. [82]	Lease	Proactive	Grid Federation	Origin	Preemption	Admission ctrl
NDDE [67]	VM	Reactive	Desktop Grid	Origin	Preemption	Local sched
Gong et al. [38]	Job	Proactive	NOW	Origin	Preemption	Local sched
Delegated-matchmaking [47]	Job	Reactive	Grid Federation	Request	Outsourcing	Outsourcing
Gruber [28]	Job	Reactive	Grid	Interdomain and Request	Global scheduling and outsourcing	Global sched and outsourcing

Shirako [48] is a lease-based platform for on-demand allocation of resources across several Clusters. In Shirako, a broker receives user's application and provides it tickets that are redeemable at the provider Cluster. In fact, Shirako brokers handles interdomain-initiated contentions by coordinating resource allocation across different Clusters. However, the user application should decide how and when to use the resources.

VioCluster [79] 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 that reacts to origin-initiated contention 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 other Cluster or waiting for nodes on a single domain.

Haizea [93] is a lease manager that is able to schedule combination of Advanced Reservation, Best Effort, and Immediate leases. Haizea acts as a scheduling back-end for OpenNebula [32]. The advantage of Haizea is considering and scheduling the preparation overhead of deploying VM disk images. For scheduling Advanced Reservation and Immediate leases, leases with lower priority (i.e., Best Effort) are preempted (i.e., suspended and resumed after the reservation is finished). In fact, Haizea provides a reactive resource contention mechanism for request-initiated contentions.

Sharc [100] is a platform that works in conjunction with nodes' operating system and enables resource sharing within Clusters. Architecturally, Sharc includes two components, namely, *control plane* and *nucleus*. The former is in charge of managing Cluster-wide resources and removing request-initiated contentions, whereas the latter, interacts with the operating system of each node and reserves resources for requests. Control plane uses a tree structure to keep information of resources are currently in use in the Cluster. The root of the tree shows all the resources in the Cluster and each child indicates one job. The nucleus uses a hierarchy that keeps information about what resources are in use on a node and by whom. The root of hierarchy shows all the resources on that node and each child represents a job on that node. In fact, there is a mapping between the control plane hierarchy and the nucleus hierarchy that helps Sharc to tolerate faults.

Cluster-on-Demand [64] (COD) 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. COD is equipped with a protocol that dynamically resizes Virtual Clusters in cooperation with middleware components. COD 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.

Cluster Reserves [10] is a resource allocation for Clusters that provides services to the clients based on the notion of service class (partitioning). This is performed by allocating resource partitions to parallel applications and dynamically adjusting the partitions on each node based on the user demand. Indeed, Cluster Reserve applies partitioning scheme to cope with the request-initiated contention problems. The resource management problem is considered as a constrained optimization problem where the inputs of the problem are periodically updated based on the resource usage.

Muse [23] is an economy-based architecture for dynamic resource procurement within a job-based Cluster. Muse is prone to request-initiated contention and applies a utility-based, economic solution to resolve that. In the model, each job has a utility function based on its throughput that reflects the revenue earned by running the job. There is a penalty that the job charges the system when its constrains are not met. Resource allocation is worked out through solving an optimization problem that maximizes the overall profit. Muse considers energy as a driving issue in resource management of server Clusters.

MUSCLE [42] is an off-line, global scheduler for multiclusters that batches parallel jobs with high packing potential (i.e., jobs that can be packed into a resource space of a given size) to the same Cluster. In the next step, a local scheduler (called TITAN) performs further tuning to run the assigned jobs with minimized make span and idle times.

Lee et al. [60] have proposed a global and a local scheduler for a multicluster. The local scheduler is a variant of backfilling that grants priority to wide jobs to decrease their waiting time and resolves the request-initiated contention. The global dispatcher assigns requests to the proper Cluster by comparing the proportion of requests with the same size at each participant Cluster. Therefore, a fairly uniform distribution of requests in the Clusters is created which leads to a considerable impact on the performance.

Percival et al. [73] applied an admission control policy for shared Cluster. There is a request-initiated contention because some large jobs takes precedence over many small jobs that are waiting in the queue. Resource providers determine the resource prices based on the degree of contention and instantaneous utilization of resources. Consumers also bid for the resources based on their budget. In general, a job can get a resource if it can compensate the loss of earning resulting from not admitting several small jobs.

57.5.2 Contention Management in Desktop Grids

This form of distributed computing (also known as volunteer computing) inherently relies on participation of resources, mainly Personal Computers. In desktop Grids, participants become available during their idle periods to leverage the execution of long running jobs. They usually use specific events such as screen-saver as an indicator for idle cycles. SETI@home [8] is a famous desktop Grid project that works based on BOINC [7] software platforms and was originally developed to explore the existence of life out of the earth. Desktop Grids are prone to origin-initiated resource contentions that take place between the guest requests (come from the Grid environment) and local requests (initiated by the resource owner) in a node.

In desktop Grids, the guest applications are running in the user (owner) environment. Running the external jobs along with other owner's processes, raised the security concern in desktop Grids and became an obstacle in prevalence of these systems. However, using the emulated platforms, such as Java, and sand-boxing the security concern were mitigated.

Another approach in desktop Grids is rebooting the machine and run an entirely independent operating system for the guest request. As a result, the guest request does not have access to the user environment. Instances of this approach include HP's I-cluster [77] and vCluster [27]. However, this approach can potentially interrupt the interactive user (owner). Therefore, idle cycle prediction has to be done conservatively to avoid interrupting the interactive user (owner). Both of these approaches are heavily dependent on the efficient predicting and harvesting of the idle cycles. Indeed, these approaches function efficiently where there are huge idle cycles.

Recently, VM technology has been used in desktop Grids. The advantages of using VMs in these environments are threefolds. First and foremost is the security that VMs provide through an isolated execution environment. Second, VMs offer more flexibility in terms of the running environment demanded by the guest application. The third benefit is that by using VMs fragmented (unused) idle cycles, such as cycles at the time of typing or other light-weight processes, can be harvested.

NDDE [67] 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. Indeed, in this system the guest and owner applications are run 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 idle process in the host operating system and, therefore, will be run instead of idle process when the system is idle. At the time the owner has a new request, the VM and all the processes belong to NDDE are preempted and changed to "ready-to-run" state.

Fine-grained cycle sharing system (FGCS) [76] runs a guest request concurrently with the local request whenever the guest process does not degrade the efficiency of the local request. However, FGCS are prone to unavailability because of the following reasons:

1. Guest jobs are killed or migrate off the resource because of a local request
2. Host suddenly discontinue contributing resource to the system

To cope with these problems, they define unavailabilities in the form a state diagram where each state is a condition that resource becomes unavailable (e.g., contention between users, and host unavailability). The authors have applied a Semi-Markov chain Process to predict the availability. The goal of this predictor engine is determining the probabilities of not transferring to unavailable states in a given time period of time in future.

57.5.3 Contention Management in Grids

Grids are initially structured based on the idea of the virtual organizations (VOs). A VO is a set of users from different organizations who collaborate towards a common objective. Several organizations constitute a VO by contributing share of their resources to that and as a result their users gain access to the VO resources. Contributing resources to a VO is carried out via an agreement upon that an organization gets access to the VO resources according to the amount of resources it offers to the VO.

Organizations usually retain part of their resources for their organizational (local) users. In other words, VO (external) requests are welcome to use resources if they are available. However, VO requests should not delay the execution of local requests.

Indeed, Grids are huge interconnected distributed systems that are prone to all kinds of resource contentions [85]. Particularly, interdomain-initiated resource contention arises when organizations need to access VO's resources based on their contributions. Origin-initiated resource contention occurs when there is a conflict between local and external users within the resources of an organization. Finally, request-initiated contention exists between different types of requests (short/long, parallel/serial, etc.).

Gruber/Di-Gruber [28] is a Grid broker that deals with the problem of resource procurement form several VOs and assigns them to different user groups. Gruber provides monitoring facilities that can be used for interdomain-initiated contentions. It also investigates the enforcing of usage policies (SLA) as well as monitoring the enforcement. Another component of Gruber sought to cope with request-initiated resource contention through monitoring resources' loads and outsource jobs to a suitable site (site selector component). Di-Gruber is the distributed version of Gruber which supports multiple decision points.

InterGrid [26] is a federation of Grid systems where each Grid receives lease requests from other Grids based on peering arrangements between InterGrid Gateways (IGG) of the Grids. Each Grid serves its own users (e.g., organizational/local users) as well as users coming from other Grids (external). InterGrid is prone to origin-initiated (between local and external requests) and interdomain-initiated (between different Grids) resource contentions.

Peering arrangements between Grids coordinate exchanging resources and functions based on peer-to-peer relations established among Grids. Each peer is built upon a predefined contract between Grids and handles interdomain-initiated contentions between the two Grids. Outsourcing unit of InterGrid is incorporated in the scheduling and determines when to outsource a request. Salehi et al. [6] have utilized probabilistic methods and proposed contention-aware scheduling that aims at minimizing the number of VM preemptions (and therefore minimizing contention) in a Grid.

They have also come up [82] with an admission control policy to reduce origin-initiated contention in InterGrid. The admission control policy works based on limiting queue length for external requests in a way that their deadline can be met. For that purpose they anticipate the average response time of external requests waiting in the queue by considering characteristics of local requests such as interarrival rate and size. In this situation, the external requests are accepted up until the response time is less than the average deadline.

Delegated-matchmaking [47] proposes an architecture that delegates the ownership of resources to users in a transparent and secure way. More specifically, when a site cannot satisfy its local users, the matchmaking mechanism of Delegated-matchmaking adds remote resources to the local resources. In fact, in Delegated-matchmaking the ownership of resources are delegated in different sites of Grids. From the resource contention perspective, matchmaking mechanism is in charge of dealing with request-initiated contentions through outsourcing scheme.

GridWay [103] is a project that creates loosely coupled connection between Grids via connecting to their meta-schedulers. GridWay is specifically useful when a job does not get the required processing power or the job waiting time is more than an appointed threshold. In these situation, GridWay migrates (outsource) the job to another Grid in order to provides the demanded resources to the job. We can consider GridWay as a global scheduler that deals with request-initiated resource contentions.

OurGrid [9] is a Grid that operates based on a P2P network between sites and share resources based on reciprocity. OurGrid uses network of favors as the resource exchange scheme between participants. According to this network, each favor to a consumer should be reciprocated by the consumer site at a later time. The more favor participants do, the more reward they expect. From the resource contention perspective, OurGrid uses incentive-based approach to figure out the problem of interdomain-initiated contentions in a Grid.

Sandholm et al. [86] investigated how admission control can increase user fulfillment in a computational market. Specifically, they considered the mixture of best effort (to improve resource utilization) and QoS-constrained requests (to improve revenue) within a virtualized Grid. They applied a reactive approach through partial preemption of best-effort requests to resolve request-initiated contentions. However, the admission control proactively accepts a new request if the QoS requirements of the current requests can still be met.

57.5.4 Contention Management in Clouds

Advances in virtual machine and network technologies has led to appearing commercial providers that offer numerous resources to users and charge them in a pay-as-you-go fashion. Since the physical infrastructure is unknown to the users in these providers; they are known as Cloud Computing [18]. There are various fashions for delivering Cloud services, which are generally known as XaaS (X as a Service). Among these services Infrastructure as a Service (IaaS) offers resources in the form of VM to users.

From the availability perspective, Cloud providers are categorized as public, private, and hybrid Clouds [19]. To cope with the shortage of resource availability, particularly in private Clouds, the idea of federated Cloud has been presented [18]. Cloud federation is a possible solution for a Cloud provider in order to access to a larger pool of resources.

Similar to Grid environments, Clouds are also prone to different types of resource contentions. However, as Clouds are more commercialized in comparison with Grids, the resource contentions solutions are also mostly commercially driven.

Recently, Amazon started to offer spot instances to sell the unused capacity of their data centers [110]. Spot instances are priced dynamically based on users' bids. If the bid price is beyond the current spot instance price, the VM instance is created for the user. The spot instance's price fluctuates and if the current price goes beyond the bid price, the VM instance is canceled (terminated) or alternatively suspended up until the current price becomes lower than the bid. Indeed, the spot market presents a request-initiated resource contention where the contention is solved via an auction-based scheme. Kondo and Andryejak [110] have evaluated the dynamic checkpointing schemes, which is adaptive to the current instance price, and achieves cost efficiency and reliability in dealing with spot instances.

Van et al. [101] have proposed a multilayer, contention-aware resource management system for Cloud infrastructure. The resource management system takes into account both request's QoS requirements and energy consumption costs in VM placement. In the request (user) level, a local decision module (LDM) monitors the performance of each request and calculates a utility function that indicates the performance satisfaction of that request. LDM interacts with a global decision module (GDM) which is the decision-making component in the architecture. GDM considers the utility functions of all LDMs along with system-level performance metrics and decides about the appropriate action. In fact, GDM provides a global scheduling solution to resolve request-initiated contentions between requests. The output of the GDM can be management commands to the server hypervisor and notifications for LDMs. The notifications for LDM includes adding a new VM to the application, upgrading or downgrading an existing VM, preempting a VM belonging to a request. Management actions for hypervisors include the starting, stopping, or live migration of a VM.

RESERVOIR [78] is a research initiative that aims at developing the technologies required to address the scalability problems existing in the single provider Cloud computing model. To achieve this goal, Clouds with excess capacity offer their resources, based on an agreed price, to the Clouds that require extra resources. Decision making about where to allocate resources for a given request is carried out through an outsourcing component, which is called placement policy. Therefore, the aim of project is providing an outsourcing solution for request-initiated resource contention.

InterCloud [18] aims to create a computing environment that offers dynamic scaling up and down capabilities (for VMs, services, storage, and database) in response to users' demand variations. The central element in InterCloud architecture is the Cloud Exchange, which is a market that gathers service providers and users' requests. It supports trading of Cloud services based on competitive economic models, such as financial options [98]. Toosi et al. [18,97] consider circumstances that each Cloud offers on-demand and spot VMs. The admission control unit evaluates the cost–benefit of outsourcing an on-demand request to the InterCloud or allocating resource to that via terminating spot VMs (request-initiated contention). Their ultimate objective is to decrease the rejection rate and having access to seemingly unlimited resources for on-demand requests.

57.6 Conclusions and Future Research Directions

Due to resource shortage as well as surge in demand, distributed systems commonly face contention between requests to access resources. Resource contentions are categorized as *request-initiated*, when a user request cannot be admitted or cannot acquire sufficient resources because the resources are occupied by other requests. *Origin-initiated* resource contention refers to circumstances that requests are from different sources with distinct priorities. *Interdomain-initiated* resource contentions take place when the proportion of shared resources to the consumed resources by a resource provider is low.

Resource contention can be handled by different components of resource management system. Therefore, solutions for resource contention depends on the structure of resource management in a distributed system. In this research, we recognized the role of resource provisioning model, local scheduling, global scheduling, and admission control unit in a resource management system on various types of resource contentions. We also realized that the emergence of VM-based resource provisioning model has posed the preemption as a predominant solution for different types of resource contentions. Therefore, in this survey we also investigated the challenges and opportunities of preempting VMs.

We reviewed systems in Clusters, Grids, and Clouds from the contention management perspective and categorized them based on their operational model, the type of contention they deal with, the component of resource management system involved in resolving the contention, and the provisioning model that contention is considered. We also closely investigated preemption mechanism as the substantial resolution for resource contention.

There are avenues of future research works in managing resource contentions that can be pursued by researchers. Proactive resource contention management methods are required specifically for interdomain-initiated contentions. This means that in an interconnected distributed system when resource management system decides to outsource a request, contention probability in the destination provider has to be considered.

Combination of different resource contentions (hybrid contention) requires further investigation. For example, resolving contention where there is a combination of origin-initiated and request-initiated contentions. Moreover, economical solutions can be taken into consideration to resolve the origin-initiated and interdomain-initiated resource contentions.

We also enumerated several options that can be considered for resuming a preempted request. Current systems usually choose one of these options. However, it will be interesting to come up with a mechanism that dynamically (e.g., based on the request condition) chooses one of the available options. A more specific case is when preemption via suspension happens. In this situation, determining the appropriate place to resume the preempted request is a challenge. For instance, if it is data-intensive request, then it might be better to wait in the queue instead of migrating to another resource.

References

1. Bright cluster manager. http://www.brightcomputing.com. AQ4
2. S. Adabala, V. Chadha, P. Chawla et al. From virtualized resources to virtual computing grids: The in-vigo system. *Future Generation Computer Systems*, 21(6):896–909, 2005.
3. L. Amar, A. Barak, E. Levy, and M. Okun. An on-line algorithm for fair-share node allocations in a cluster. In *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid*, pp. 83–91, Rio de Janeiro, Brazil, 2007.
4. L. Amar, A. Mu'Alem, and J. Stober. The power of preemption in economic online markets. In *Proceedings of the 5th International Workshop on Grid Economics and Business Models (GECON'08)*, pp. 41–57, Berlin, Germany, 2008.
5. M.A. Salehi, B. Javadi, and R. Buyya. Resource provisioning based on leases preemption in InterGrid. In *Proceeding of the 34th Australasian Computer Science Conference (ACSC'11)*, pp. 25–34, Perth, Australia, 2011. AQ5
6. M.A. Salehi, B. Javadi, and R. Buyya. QoS and preemption aware scheduling in federated and virtualized grid computing environments. *Journal of Parallel and Distributed Computing (JPDC)*, 72(2):231–245, 2012.
7. D.P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of 5th IEEE/ACM International Workshop on Grid Computing*, pp. 4–10, 2004.

8. D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@ home: An experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.
9. N. Andrade, F. Brasileiro, W. Cirne, and M. Mowbray. Automatic grid assembly by promoting collaboration in peer-to-peer grids. *Journal of Parallel and Distributed Computing*, 67(8):957–966, 2007.
10. M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Proceedings of the International Conference on Measurement and Modelling of Computer Systems (SIGMETRICS'00)*, pp. 90–101, 2000.
11. M.D. Assunção and R. Buyya. Performance analysis of multiple site resource provisioning: Effects of the precision of availability information. In *Proceedings of the 15th International Conference on High Performance Computing (HiPC'08)*, pp. 157–168, 2008.
12. M.D. Assunção, A.D. Costanzo, and R. Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Proceedings of the 19th International Symposium on High Performance Distributed Computing (HPDC09)*, pp. 141–150, 2009.
13. A. Barak, A. Shiloh, and L. Amar. An organizational grid of federated mosix clusters. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid05)*, vol. 1, pp. 350–357, 2005.
14. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.
15. P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh. Spruce: A system for supporting urgent high-performance computing. In *Grid-Based Problem Solving Environments*, pp. 295–311, 2007.
16. B. Beeson, S. Melniko, S. Venugopal, and D.G. Barnes. A portal for grid-enabled physics. In *Proceeding of the 28th Australasian Computer Science Week (ACSW'05)*, pp. 13–20, 2005.
17. R. Bolze, F. Cappello, E. Caron et al. Grid'5000: A large scale and highly reconfigurable experimental Grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481, 2006.
18. R. Buyya, R. Ranjan, and R.N. Calheiros. InterCloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing-Volume Part I*, pp. 13–31, 2010.
19. R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.
20. K. Chanchio, C. Leangsuksun, H. Ong, V. Ratanasamoot, and A. Shafi. An efficient virtual machine checkpointing mechanism for hypervisor-based HPC systems. In *Proceeding of the High Availability and Performance Computing Workshop (HAPCW)*, pp. 29–35, 2008.
21. K. Chanchio and X.H. Sun. Data collection and restoration for heterogeneous process migration. *Software: Practice and Experience*, 32(9):845–871, 2002.
22. K.M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 3(1):63–75, 1985.
23. J.S. Chase, D.C. Anderson, P.N. Thakar, A.M. Vahdat, and R.P. Doyle. Managing energy and server resources in hosting centers. *ACM SIGOPS Operating Systems Review*, 35(5):103–116, 2001.
24. C. Coti, T. Herault, P. Lemarinier, L. Pilard, A. Rezmerita, E. Rodriguez, and F. Cappello. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant MPI. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pp. 127–1233, 2006.
25. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing*, pp. 181–194. IEEE, 2001.
26. M.D. De Assunção, R. Buyya, and S. Venugopal. InterGrid: A case for internetworking islands of Grids. *Concurrency and Computation: Practice and Experience*, 20(8):997–1024, 2008.
27. C. De Rose, F. Blanco, N. Maillard, K. Saikoski, R. Novaes, O. Richard, and B. Richard. The virtual cluster: A dynamic network environment for exploitation of idle resources. In *Proceedings 14th Symposium on Computer Architecture and High Performance Computing*, pp. 141–148. IEEE, 2002.

28. C. Dumitrescu, I. Raicu, and I. Foster. Di-gruber: A distributed approach to grid resource brokering. In *Proceedings of ACM/IEEE Conference on Supercomputing*, pp. 38–45, 2005.
29. W. Emeneker, D. Jackson, J. Butikofer, and D. Stanzione. Dynamic virtual clustering with xen and moab. In *Frontiers of High Performance Computing and Networking–ISPA Workshops*, pp. 440–451. Springer, 2006.
30. A. Ferrari, S.J. Chapin, and A. Grimshaw. Heterogeneous process state capture and recovery through process introspection. *Cluster Computing*, 3(2):63–73, 2000.
31. L. Field and M. Schulz. Grid interoperability: The interoperations cookbook. *Journal of Physics: Conference Series*, 119:120–139. IOP Publishing, 2008.
32. J. Fontán, T. Vázquez, L. Gonzalez, R.S. Montero, and I.M. Llorente. OpenNebula: The open source virtual machine manager for cluster computing. In *Open Source Grid and Cluster Software Conference*, San Francisco, CA, May 2008.
33. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115–128, 1997.
34. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.
35. S. Gabriel. Gridka tier1 site management. In *International Conference on Computing in High Energy and Nuclear Physics (CHEP'07)*, 2007.
36. S. Garg, S. Venugopal, and R. Buyya. A meta-scheduler with auction based resource allocation for global grids. In *14th IEEE International Conference on Parallel and Distributed Systems (ICPADS'08)*, pp. 187–194, 2008.
37. S. Garg, C. Yeo, A. Anandasivam, and R. Buyya. Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71(6):732–749, 2011.
38. L. Gong, X. Sun, and E.F. Watson. Performance modeling and prediction of nondedicated network computing. *IEEE Transactions on Computers*, 51(9):1041–1055, 2002.
39. L. Grit, L. Ramakrishnan, and J. Chase. On the duality of jobs and leases. Technical report CS-2007-00, Duke University, Department of Computer Science, April 2007.
40. I. Grudenić and N. Bogunović. Analysis of scheduling algorithms for computer clusters. In *Proceeding of 31th International Convention on Information and Communication Technology. Electronics and Microelectronics (MIPRO)*, pp. 13–20, 2008.
41. P.H. Hargrove and J.C. Duell. Berkeley lab checkpoint/restart (BLCR) for linux clusters. *Journal of Physics: Conference Series*, 46:494. IOP Publishing, 2006.
42. L. He, S.A. Jarvis, D.P. Spooner, X. Chen, and G.R. Nudd. Dynamic scheduling of parallel jobs with QoS demands in multiclusters and grids. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pp. 402–409. IEEE Computer Society, 2004.
43. J. Heo, S. Yi, Y. Cho, J. Hong, and S.Y. Shin. Space-efficient page-level incremental checkpointing. In *Proceedings of the ACM Symposium on Applied Computing*, pp. 1558–1562, 2005.
44. F. Hermenier, A. Lèbre, and J. Menaud. Cluster-wide context switch of virtualized jobs. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*, New York, pp. 658–666, 2010.
45. M. Hovestadt, O. Kao, A. Keller, and A. Streit. Scheduling in HPC resource management systems: Queuing vs. planning. In *Proceedings of 9th International Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 1–20, 2003.
46. W. Huang, Q. Gao, J. Liu, and D.K. Panda. High performance virtual machine migration with RDMA over modern interconnects. In *Proceedings of the 9th IEEE International Conference on Cluster Computing*, pp. 11–20. IEEE, 2007.
47. A. Iosup, O. Sonmez, S. Anoep, and D. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, pp. 97–108, 2008.

48. D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K.G. Yocum. Sharing networked resources with brokered leases. In *USENIX Annual Technical Conference*, pp. 199–212, Boston, MA, June 2006.
49. M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *Proceedings of the 22nd Symposium on Operating Systems Principles (SOSP), ACM SIGOPS*, pp. 261–276, 2009.
50. S.T. Jones, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau. Antfarm: Tracking processes in a virtual machine environment. In *Proceedings of the USENIX Annual Technical Conference*, pp. 1–14, 2006.
51. A. Kansal, F. Zhao, J. Liu, N. Kothari, and A.A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, pp. 39–50, 2010.
52. K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. Science clouds: Early experiences in cloud computing for scientific applications. In *Proceeding of Cloud Computing and Applications*, vol. 1, 2008.
53. K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual workspaces in the grid. In *Proceeding of the 11th International Euro-Par Conference on Parallel Processing*, pp. 421–431, 2005.
54. R. Kettimuthu, V. Subramani, S. Srinivasan, T. Gopalsamy, D.K. Panda, and P. Sadayappan. Selective preemption strategies for parallel job scheduling. *International Journal of High Performance and Networking (IJHPCN)*, 3(2/3):122–152, 2005.
55. A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: The linux virtual machine monitor. In *Linux Symposium*, p. 225, 2007.
56. I. Krsul, A. Ganguly, J. Zhang, J.A.B. Fortes, and R.J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *Proceedings of the ACM/IEEE Conference Supercomputing*, pp. 7–17, 2004.
57. Y. Kwok and I. Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Transaction Parallel and Distributed Systems*, 7(5):506–521, 1996.
58. Y. Kwok, S.S. Song, K. Hwang et al. Selfish grid computing: Game-theoretic modeling and nas performance results. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, vol. 2, pp. 1143–1150, 2005.
59. B.G. Lawson and E. Smirni. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. *ACM SIGMETRICS Performance Evaluation Review*, 29(4):40–47, 2002.
60. H.Y. Lee, D.W. Lee, and R. Ramakrishna. An enhanced grid scheduling with job priority and equitable interval job distribution. In *Advances in Grid and Pervasive Computing*, pp. 53–62, 2006.
61. H. Li, D.L. Groep, and L. Wolters. Workload characteristics of a multi-cluster supercomputer. In *Proceedings of 10th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'04)*, pp. 176–193, 2004.
62. M. Litzkow and M. Solomon. The evolution of condor checkpointing. In *Mobility: Processes, Computers, and Agents*, pp. 163–174. ACM Press/Addison-Wesley Publishing Co., 1999.
63. M. Margo, K. Yoshimoto, P. Kovatch, and P. Andrews. Impact of reservations on production job scheduling. In *Proceedings of 13th International Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 116–131. Springer, 2008.
64. J. Moore, D. Irwin, L. Grit, S. Sprenkle, and J. Chase. Managing mixed-use clusters with cluster-on-demand. Technical report, Duke University Department of Computer Science, 2002.
65. H. Nakada, H. Sato, K. Saga, M. Hatanaka, Y. Saeki, and S. Matsuoka. Job invocation interoperability between naregi middleware beta and glite. In *Proceedings of the 9th International Conference on High Performance Computing, Grid and e-Science in Asia Pacific Region (HPC Asia 2007)*, 2007.
66. R. Nathuji and K. Schwan. Virtualpower: Coordinated power management in virtualized enterprise systems. *ACM SIGOPS Operating Systems Review*, 41(6):265–278, 2007.
67. R.C. Novaes, P. Roisenberg, R. Scheer, C. Northfleet, J.H. Jornada, and W. Cirne. Non-dedicated distributed environment: A solution for safe and continuous exploitation of idle cycles. *Scalable Computing: Practice and Experience*, 6(3), 2001.

AQ6

68. D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09)*, pp. 124–131, 2009.
69. H. Ong, N. Saragol, K. Chanchio, and C. Leangsuksun. Vccp: A transparent, coordinated checkpointing system for virtualization-based cluster computing. In *IEEE International Conference on Cluster Computing and Workshops, (CLUSTER'09)*, pp. 1–10. IEEE, 2009.
70. S. Ontañón and E. Plaza. Cooperative case bartering for case-based reasoning agents. In *Topics in Artificial Intelligence*, pp. 294–308, 2002.
71. A. Oram, ed. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates Inc., Sebastopol, CA, 2001.
72. E. Parsons and K. Sevcik. Implementing multiprocessor scheduling disciplines. In *Proceedings of 3rd International Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 166–192, 1997.
73. X. Percival, C. Wentong, and L. Bu-Sung. A dynamic admission control scheme to manage contention on shared computing resources. *Concurrency and Computing: Practice and Experience*, 21:133–158, February 2009.
74. J.S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent checkpointing under unix. In *Proceedings of the USENIX Technical Conference*, pp. 18–28. USENIX Association, 1995.
75. I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde. Falkon: A fast and light-weight task execution framework. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'07)*, pp. 1–12. IEEE, 2007.
76. X. Ren, S. Lee, R. Eigenmann, and S. Bagchi. Prediction of resource availability in fine-grained cycle sharing systems empirical evaluation. *Journal of Grid Computing*, 5(2):173–195, 2007.
77. B. Richard and P. Augerat. I-cluster: Intense computing with untapped resources. In *Proceeding of 4th International Conference on Massively Parallel Computing Systems*, pp. 127–140, 2002.
78. B. Rochwerger, D. Breitgand, A. Epstein et al. Reservoir-when one cloud is not enough. *Computer Journal*, 44(3):44–51, 2011.
79. P. Ruth, P. McGachey, and D. Xu. Viocluster: Virtualization for dynamic computational domain. In *Proceedings of International IEEE Conference on Cluster Computing (Cluster'05)*, pp. 1–10, Burlington, September 2005.
80. F.D. Sacerdoti, M.J. Katz, M.L. Massie, and D.E. Culler. Wide area cluster monitoring with ganglia. In *Proceedings of International Conference on Cluster Computing*, pp. 289–298. IEEE, 2003.
81. M.A. Salehi, B. Javadi, and R. Buyya. Performance analysis of preemption-aware scheduling in multi-cluster grid environments. In *Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'11)*, pp. 419–432, 2011.
82. M.A. Salehi, B. Javadi, and R. Buyya. Preemption-aware admission control in a virtualized grid federation. In *Proceeding of 26th International Conference on Advanced Information Networking and Applications (AINA'12)*, pp. 854–861, 2012.
83. M.A. Salehi, P.R. Krishna, K.S. Deepak, and R. Buyya. Preemption-aware energy management in virtualized datacenters. In *Proceeding of 5th International Conference on Cloud Computing (IEEE Coud'12)*, 2012.
84. M.A. Salehi and R. Buyya. Adapting market-oriented scheduling policies for cloud computing. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing-Volume Part I*, pp. 351–362. Springer-Verlag, 2010.
85. M.A. Salehi, H. Deldari, and B.M. Dorri. Balancing load in a computational grid applying adaptive, intelligent colonies of ants. *Informatica: An International Journal of Computing and Informatics*, 33(2):151–159, 2009.
86. T. Sandholm, K. Lai, and S. Clearwater. Admission control in a computational market. In *Eighth IEEE International Symposium on Cluster Computing and the Grid*, pp. 277–286, 2008.
87. H. Shachnai, T. Tamir, and G.J. Woeginger. Minimizing makespan and preemption costs on a system of uniform machines. *Algorithmica Journal*, 42(3):309–334, 2005.

88. P.C. Shih, H.M. Chen, Y.C. Chung, C.M. Wang, R.S. Chang, C.H. Hsu, K.C. Huang, and C.T. Yang. Middleware of taiwan unigrid. In *Proceedings of the ACM Symposium on Applied Computing*, pp. 489–493. ACM, 2008.
89. M. Silberstein, D. Geiger, A. Schuster, and M. Livny. Scheduling mixed workloads in multi-grids: The grid execution hierarchy. In *Proceedings of 15th Symposium on High Performance Distributed Computing*, pp. 291–302, 2006.
90. W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, pp. 127–132. IEEE, 2000.
91. Q. Snell, M.J. Clement, and D.B. Jackson. Preemption based backfill. In *Proceedings of 8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pp. 24–37. Springer, 2002.
92. A. Sodan. Service control with the preemptive parallel job scheduler scojo-pect. *Journal of Cluster Computing*, pp. 1–18, 2010. AQ7
93. B. Sotomayor, K. Keahey, and I. Foster. Combining batch execution and leasing using virtual machines. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, pp. 87–96, New York, 2008. ACM.
94. B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster. Resource leasing and the art of suspending virtual machines. In *Proceedings of the 11th IEEE International Conference on High Performance Computing and Communications*, Washington, DC, pp. 59–68, 2009.
95. G. Stellner. Cocheck: Checkpointing and process migration for mpi. In *Proceedings of the the 10th International Parallel Processing Symposium*, pp. 526–531, 1996.
96. D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 17(2–4):323–356, 2005.
97. A.N. Toosi, R.N. Calheiros, R.K. Thulasiram, and R. Buyya. Resource provisioning policies to increase iaas provider's profit in a federated cloud environment. In *Proceeding of 13th International Conference on High Performance Computing and Communications (HPCC)*, pp. 279–287, 2011.
98. A.N. Toosi, R.K. Thulasiram, and R. Buyya. Financial option market model for federated cloud environments. Technical report, Department of Computing and Information System, Melbourne University, 2012.
99. D. Tsafrir, Y. Etsion, and D.G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):789–803, 2007.
100. B. Urgaonkar and P. Shenoy. Sharc: Managing cpu and network bandwidth in shared clusters. *IEEE Transactions on Parallel and Distributed Systems*, 15(1):2–17, 2004.
101. H.N. Van, F.D. Tran, and J.M. Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Proceedings of the 9th Conference on Computer and Information Technology, vol. 02*, pp. 357–362. IEEE Computer Society, 2009.
102. C. Vázquez, E. Huedo, R.S. Montero, and I.M. Llorente. Federation of TeraGrid, EGEE and OSG infrastructures through a metascheduler. *Future Generation Computing Systems*, 26(7):979–985, 2010.
103. J.L. Vázquez-Poletti, E. Huedo, R.S. Montero, and I.M. Llorente. A comparison between two grid scheduling philosophies: EGEE WMS and grid way. *Multiagent Grid Systems*, 3:429–439, December 2007.
104. S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling e-science applications on global data grids. *Concurrency and Computation: Practice and Experience*, 18(6):685–699, 2006.
105. A. Verma, P. Ahuja, and A. Neogi. pMapper: Power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pp. 243–264. Springer-Verlag, New York, Inc., 2008.
106. A. Verma, P. Ahuja, and A. Neogi. Power-aware dynamic placement of HPC applications. In *Proceedings of the 22nd Annual International Conference on Supercomputing*, pp. 175–184, 2008.
107. W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. *Cloud Computing*, pp. 254–265, 2009.

108. E. Walker, J.P. Gardner, V. Litvin, and E.L. Turner. Creating personal adaptive clusters for managing scientific jobs in a distributed computing environment. In *Proceedings of the Challenges of Large Applications in Distributed Environments (CLADE'06)*, pp. 95–103. IEEE, 2006.
109. J.P. Walters, B. Bantwal, and V. Chaudhary. Enabling interactive jobs in virtualized data centers. *Cloud Computing and Applications*, 2008. AQ8
110. S. Yi, D. Kondo, and A. Andrzejak. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *Proceedings of 3rd IEEE International Conference on Cloud Computing (CLOUD)*, pp. 236–243, 2010.
111. H. Zhong and J. Nieh. Crak: Linux checkpoint/restart as a kernel module. Technical report, Department of Computer Science, Columbia University, New York, 2001.
112. M. Zwahlen. Managing contention in collaborative resource sharing systems using token-exchange mechanism. PhD thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2007.

Author Queries

[AQ1] Please check part caption b for sense.
[AQ2] Would you like to explain the abbreviations "sched" and "ctrl" as a footnote? Please check.
[AQ3] Please check if this should be "Fairness and preemption"
[AQ4] Please provide the complete details for Ref. [1].
[AQ5] Please provide the location details for Refs. [5,10–15,18,20,24,25,27–29,36,39,40,42,45,46, 49–53,55,56,60–65,68,69,72,74,75,77,79,80–84,86,88–91,95,97,98,101,106,108,110].
[AQ6] Please provide the page range for Ref. [67].
[AQ7] Please provide the volume number for Refs. [92,107].
[AQ8] Please provide the complete reference details for Ref. [109].