Contention-aware Resource Provisioning in Interconnected Grid Computing Systems

by

Mohsen Amini Salehi

Submitted in total fulfilment of
the requirements for the degree of

Doctor of Philosophy

Department of Computing and Information Systems
The University of Melbourne, Australia

August 2012


ای که با نامت جهان آغاز شد
دفتر ما هم به نامت باز شد
دفتری کز نام تو زینت گرفت
کار آن از چرخ بالاتر گرفت

Dedicated to those who enlightened my life.

Contention-aware Resource Provisioning in Interconnected Grid Computing Systems

Mohsen Amini Salehi
Principal Supervisor: Prof. Rajkumar Buyya

Abstract

Resource sharing environments enable sharing and aggregation of resources across several resource providers. InterGrid provides an architecture for resource sharing based on virtual machine technology between Grids. Resource providers in InterGrid serve their local requests as well as external requests assigned by InterGrid. However, resource providers would like to ensure that the requirements of their local requests are not delayed because of running external requests. This scenario leads to contention for resources between the external and local requests.

In this dissertation, preemption mechanism is considered to resolve the contention, while side-effects of this mechanism are taken into account. Particularly, the number of preempted external requests, their waiting time, and imposed overhead of preemption are considered. Therefore, this dissertation investigates and categorises mechanisms for management of resource contention in the existing systems. Then, it presents a contention management scheme that includes two main strategies. The first strategy avoids the contentious situation by establishing contention-awareness in the scheduling policies. The second strategy, handles contention side-effects while considering long waiting time and energy consumption issues. These strategies are proposed within different architectural elements of the InterGrid platform.

In this dissertation, first feasibility of the preemption mechanism to resolve resource contention is presented, then overhead time imposed for performing various preemption scenarios are modelled, and different policies to minimise the side-effects of resource contention are proposed. To avoid resource contention, a scheduling policy is proposed in gateway (meta-scheduling) level, that proactively disseminates external requests on resource providers. Also, a dispatch policy is proposed to decrease the likelihood of resource contention for more valuable external users. To prevent long waiting time for external requests, an admission control policy is proposed to limit the number of accepted external requests when there is a surge in demand. Then, a contention-aware energy management policy is proposed to adapt energy consumption of resource providers to user demand. This policy is for situation that resource providers operate at low utilisation and it considers long waiting time for external requests.

Performance evaluations of the strategies are achieved using discrete-event simulation. This dissertation also realises the proposed scheme in InterGrid.

Declaration

This is to certify that:

(i) the thesis comprises only my original work towards the PhD except where indicated,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of table, maps, bibliographies, appendices and footnotes.

Mohsen Amini Salehi

Acknowledgements

PhD is a long and amazing journey all about *learning*. Many people appear in the path of this journey who enrich your knowledge and life. To all of them, I would like to extend my most heartfelt thanks:

First and foremost, to Professor Rajkumar Buyya, my principal supervisor for his intellectual guidance, giving me the opportunity to think freely on my research, and his continuous support during and after PhD. His endless motivation has always inspired me.

To Associate Professor Jemal Abawajy, for his advice throughout my candidature. To Dr. Bahman Javadi for his friendly manner, constructive comments, and providing me with encouragement during rough times. To Adel Nadjarran Toosi whose companionship and encouragements improved both my research and life. To William Voorsluys, who taught me far beyond what matters in PhD. To Dr. Rodrigo Calheiros who was kind to read the dissertation word by word. To all past and current members of the CLOUDS Laboratory in Melbourne University. In particular, Amir Vahid, Saurabh Garg, Dr. Marcos Dias de Assuncao, Anton Beloglazov, Atefe Khosravi, Sare Fotouhi, Nikolay Grozev, Deepak Poola, Sivaram Yoganathan, Linlin Wu, Mohammed Alrokayan, Dr. Marco Netto, Dr. Mustafizur Rahman, Dr. Mukaddim Pathan, Dr. Suraj Pandey, Dr. Rajiv Ranjan, Dr. Christian Vecchiola, Dr. Srikumar Venugopal for their helpful comments. To the staff and friends from the CIS Department for their support, specially Mani Abedini, Saeed Shahbazi, and Masud Moshtaghi. To the University of Melbourne and Australian Research Council (ARC) for financial support of my candidature.

I would like to express gratitude to Associate Professor Hossain Deldari and Assistant Professor Saeid Abrishami my Master and Bachelor degree supervisors who paved the way of my education in the Melbourne University. Thanks to Infosys Company for hosting part of my PhD. I would like to express my gratitude especially to Dr. Radha Krishna Pisipati, Krishnamurty Sai Deepak, and Inchara Shanthappa. I am also immensely grateful to my family. My wife who patiently borne with the student life and my parents who have provided me with lessons on honesty and ethics that no university could ever rivals. Thanks to my sisters for their love and support. Thanks to my Iranian friends: Afshin, Ali, Alireza, Davood, Hoda, Khadije, Mahsa, Naghme, Rozita, and Samira, who relief the pain of being far from home and make it a joyful journey. Finally, I thank God for blessing me all these opportunities and a twin lovely girls who sweeten my life.

Mohsen Amini Salehi
August 2012

Contents

List of Tables

List of Figures

Chapter 1

Introduction

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 [1] in a way that no single system (specially a single Cluster or Grid) can meet the needs of all users. Therefore, the notion of resource sharing between interconnected distributed systems has emerged [2].

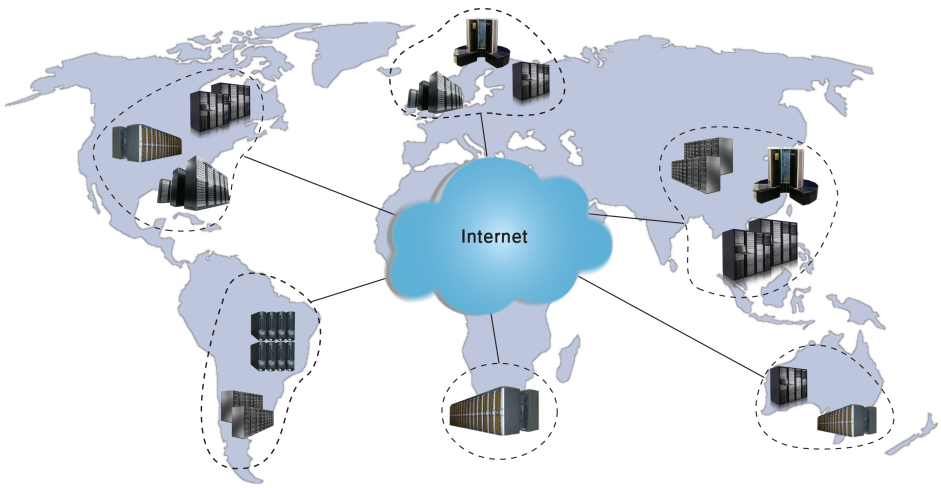


Figure 1.1: An interconnected distributed system.

In an interconnected distributed system, as depicted in Figure 1.1, organisations (also termed resource providers in this dissertation) share their resources over the Internet. The advantage of these resource sharing environments is twofold: Users can access larger resources; resource providers (hereafter termed RPs) can increase their resource utilisation by accepting requests from other systems (i.e., external requests) in addition to their local requests.

However, sharing resources with external users from other organisations can potentially affect the performance demanded by local users of an RP. Therefore,

resource providers in interconnected distributed systems, particularly in Grid computing, would like to ensure that their community has priority access to resources [3–7]. 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 circumstances that there is a surge in demand, local and external requests contend to gain access to resources over the same time period. This condition is generally known as *resource contention* between requests.

The problem this dissertation addresses is, broadly speaking, "how to resolve resource contention in an interconnected distributed system". When these systems provide distinct priorities for different users, as they almost always do, it has to be determined who gets the resources and when. Such decisions can be driven by the priority of various users or the requests'Quality of Service (QoS) requirements (e.g., deadline).

Preemption mechanism is a common solution for contentious situation and used by various resource management systems [6–11]. This mechanism stops the running request and frees the resources for another, possibly higher priority, request. Specifically, preemption is a promising mechanism due to the prevalence of Virtual Machine (VM) technology as the execution unit in existing distributed systems [12, 13]. Utilising VMs facilitates resumption of the preempted request from the preempted point.

InterGrid [1] provides an architecture for interconnecting Grids. Using InterGrid each RP receives requests from other Grids (termed external requests), while serves its local users'requests. Local and external requests contend to gain access to computational resources in an RP.

This dissertation concentrates on the resource contention between requests in InterGrid and investigates mechanisms and techniques to resolve them. The remaining part of this chapter provides a bird's-eye view of the research works presented in this dissertation, including the essence of resolving contention in InterGrid, the research problem, the objectives of this dissertation, its contributions, and its organisation.

1.1 Motivations

The main advantage of resource sharing between Grids through platforms such as InterGrid is to enable RPs to extend their capacity beyond their domain-level infrastructure limits and utilise the computing power of other providers to run

resource-intensive applications. From the resource utilisation perspective, RPs can increase their resource utilisation by accepting requests from other systems (external requests) in addition to their local requests.

The architecture of InterGrid, as depicted in Figure 1.2, is based on InterGrid Gateways (IGGs) that coordinate resource acquisition across Grids. Similar to the Internet Service Providers (ISPs) that establish peering arrangements with each other, InterGrid establishes peering between Grids using IGGs. The peering arrangements define the conditions upon which resources are shared between Grids.

In InterGrid, resource provisioning is achieved based on *lease* abstraction, where leases are implemented based on VMs. A lease in InterGrid is an agreement between a resource provider and a resource consumer whereby the provider agrees to allocate resources to the consumer according to the lease terms presented by the consumer [13]. InterGrid creates one lease for each user request. VMs have numerous advantages that make them suitable for implementing lease-based resource provisioning. Most importantly, VMs enable partitioning of a single physical machine into several virtual machines. In addition to that, VMs have their own software stack which can provide various execution environments for different users on a single physical machine. Finally, the VMs'ability to transparently suspend, resume, and migrate [12–14] without affecting the computation inside them [15] enables the scheduler to implement efficient scheduling strategies.

In InterGrid, each Grid is composed of several RPs where the resources are generally Clusters of computers managed by queue-based resource management systems [1]. These RPs contribute resources to InterGrid, while need to respect their local users'demands. Local requests in the RPs of InterGrid have priority to access resources over external requests. This scenario leads to contention for resources between external users and the RPs local users. To enable resource sharing in InterGrid, the resource contention between local and external users has to be resolved. Thus, mechanisms applied in InterGrid have to be aware of these contentions and provide techniques to efficiently handle them.

One approach to resolve the contention is partitioning of the resources in RPs and dedicating a part of them to external requests [3]. However, this approach leads to inefficient resource allocation within the RPs [16]. Another common approach is to preempt external requests in favour of local requests. Unlike the former approach, it is demonstrated that preemption mechanism reinforces dynamic resource provisioning and utilises resources more efficiently [17], specifically when the resources are provisioned using VMs.

Nonetheless, preempting VM-based leases involves 2 main side-effects. The

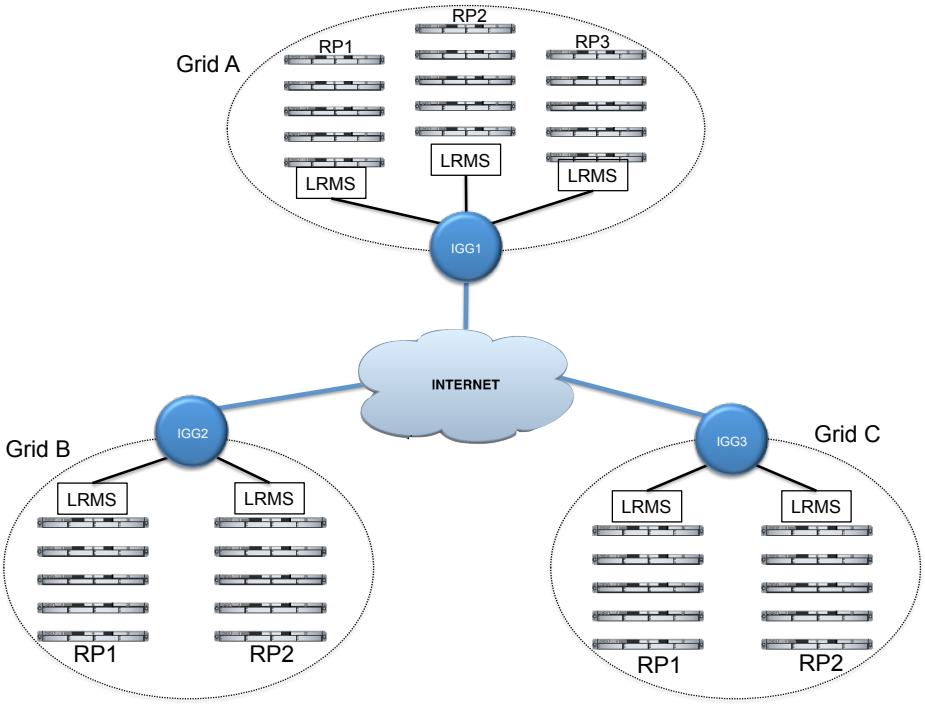


Figure 1.2: High level view of InterGrid architecture.

first side-effect is the time overhead imposed on the system for preempting VMs. The overhead varies based on the type of operation performed on the VMs. For instance, the overhead of suspending a VM is lower than the overhead of migrating it. The second side-effect, is increase in the response time of external requests due to preemption and delay in their execution.

This dissertation resolves the contention between requests in InterGrid based on the preemption mechanism. To achieve that, this dissertation investigates strategies that reduce the resource contention as well as its side-effects.

1.2 Research Problem and Objectives

This dissertation tackles the challenge of how resource contention between users'requests can be resolved in a federated Grid, in particular InterGrid, without disrespecting local users and without long response time for external users.

Towards that end, this dissertation considers the preemption mechanism to resolve the resource contention in InterGrid, as a platform for Grid federation. However, preemption involves side-effects in terms of the overhead time and long response time for low priority requests. Mechanisms applied in the InterGrid platform have to be aware of the contention as well as the side-effects of resolving it using the preemption mechanism. To efficiently handle the resource contention and its side-effects, this dissertation investigates two main strategies:

The first strategy avoids the contentious situation by establishing contention-awareness in the scheduling of users'requests. Specifically, in this strategy schedulers are aware of the contentions that can take place and try to avoid that through efficient scheduling of the requests. This strategy involves the contention-awareness in the local scheduling of the RP and in the global scheduling (gateway) level.

The second strategy handles resource contention in a way that its impact on the response time of external (low priority) requests is minimised. More specifically, the second strategy investigates the waiting time of external requests in two circumstances: First, when there is a surge in demand from local users; second, when energy management mechanisms are applied within resource providers.

1.2.1 Objectives

Based on the described challenges, we delineated the following objectives:

1. Model the imposed overhead of VM preemption, the mechanism of resolving contention in this dissertation, and investigate solutions to reduce the imposed overhead of resolving contention.

2. Investigate proactive scheduling techniques to avoid contentious situation that can arise in InterGrid.

3. Explore techniques to handle side-effects of the preemption mechanism, mainly in terms of the long response time for external requests.

1.2.2 Evaluation Methodology

Evaluation of the proposed mechanisms in this dissertation is carried out through discrete-event simulation. Simulation enables us to control the experiments and conduct them with different parameters to examine the behaviour of the mechanisms in various circumstances. Reproducibility of the environment as well as the results are other advantages of simulation-based evaluations. Additionally, it is difficult to change the configuration of computing resources in production environments to evaluate the efficacy of different mechanisms.

We implemented and evaluated contention-aware policies in the InterGrid platform to resolve resource contention between local and external requests.

1.3 Contributions

Considering the aforementioned objectives, this dissertation makes the following contributions:

1. It introduces preemption of external leases in favour of local requests as a solution for resource contention between them in InterGrid. Later, it deals with the side-effects of preempting VM-based leases, specifically, the overhead time, number of resource contentions, and response time. Therefore, a model for estimating the overhead time of preempting VMs, based on possible operations on them, is proposed. Additionally, several policies that determine the proper set of leases for preemption are proposed in the local scheduler of RPs. These policies sought to either decrease the preemption overhead or increase the user satisfaction by reducing the number of resource contentions. A third policy handles the trade-off between the overhead and the number of resource contentions. Simulation results demonstrate that the proposed preemption policies can serve local requests without increasing the rejection of external requests.

2. It investigates how resource contention can be avoided in a Grid with multiple RPs in the global scheduling (gateway) level. For this purpose, a preemption-aware workload allocation policy in the IGG is proposed to proactively distribute external requests amongst RPs in a way that the overall number of resource contentions decreases.

   Additionally, a situation that some external requests are more valuable (i.e., external requests have different QoS levels) is studied. The proposed scheduling policy investigates the impact of proactive dispatching of external requests to the RPs on reducing the probability of contention for valuable external requests. Experiment results indicate that the workload allocation policy, specifically when it is combined with the proposed dispatch policy, significantly decreases the number of resource contentions. This decrease improves the resource utilisation as well as average weighted waiting time of external requests.

3. It presents a mechanism for admission control within RPs to prevent the long response time for external requests. In fact, resource owners are interested in accepting as many external requests as possible in order to maximise their profit. However, accepting many external requests increases the likelihood of resource contention and preemption, particularly when there is a surge in demand for local requests. This situation leads to unpredictable response time for external requests.

The admission control mechanism provides a predictable system by limiting the number of accepted external requests to assure that they can be completed with a definite waiting time. An analytical queuing model is applied to find out the ideal number of external requests in an RP. Then, the preemption-aware admission control policy is derived based on the proposed model.

Experiment results indicate that the admission control mechanism significantly reduces long response times and leads to completing more external requests comparing to other similar mechanisms.

4. It provides a contention-aware energy management mechanism in RPs. As mentioned earlier, the admission control mechanism is useful when there is a surge in demand within an RP. On the contrary, when an RP operates at low utilisation, there is a potential to reduce the energy consumption by deactivating lightly loaded resources. However, decreasing the number of active resources results in more resource contention and long response time for external requests.

   Therefore, an adaptive contention-aware energy management mechanism is proposed in the RP. It adapts the energy consumption based on the performance demands. For allocating a given request, the mechanism decides about switching on, preempting, consolidating, or combination of these operations. To avoid the long response time for external requests, the energy management mechanism takes into account the waiting time of external requests.

5. The realisation of the contention-aware scheduling in InterGrid is presented. The aim is to resolve resource contention between external and local requests in the InterGrid platform that uses VM-based leases for resource provisioning. The implementation enables RPs to increase their resource utilisation through contributing resources to InterGrid and without delaying their local users. In addition to that, several contention-aware scheduling policies are implemented and evaluated in this environment that consider the amount of resource contention and imposed overhead of preemption.

1.4 Thesis Organisation

The core chapters of this dissertation are derived from several research papers published during the course of the PhD candidature. The interrelationship between chapters and the strategy to which they are related to are depicted in

Figure 1.3. The remaining part of this dissertation is organised as follows:

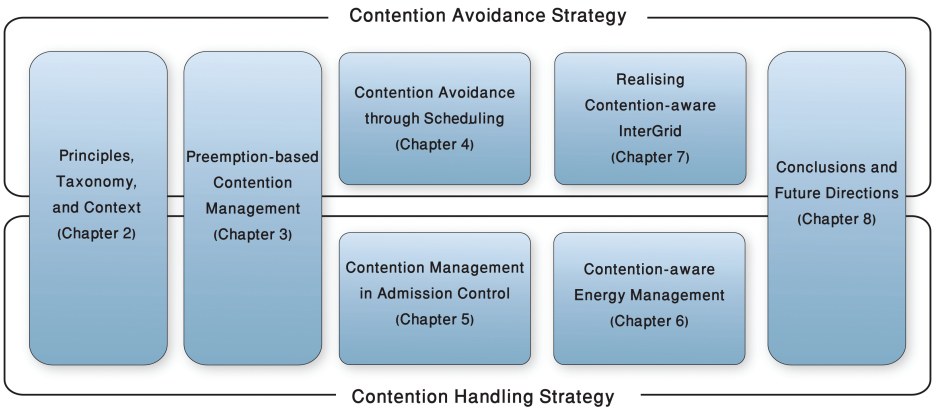


Figure 1.3: Organisation of this dissertation.

- Chapter 2 provides a literature review on the current related works in distributed computing systems area and positions our work in comparison with them. Chapter 2 derived from:
  - **M. Amini Salehi**, J. Abawajy, and R. Buyya, *Taxonomy of Contention Management in Interconnected Distributed Computing Systems*, Computing Handbook Set, CRC Press, 2012 (In Print).
- In Chapter 3, we discuss overheads involved in preempting leases in a resource provider of InterGrid. In addition to that, we propose preemption policies in the local scheduler level that are aware of preemption side-effects and try to alleviate them. Chapter 3 derived from:
  - **M. Amini Salehi**, B. Javadi, and R. Buyya, *Resource Provisioning Policies in Interconnected Virtualised Grids based on Lease Preemption*. Submitted to Journal of Concurrency and Computation: Practice and Experience (CCPE), 2012.
  - **M. Amini Salehi**, B. Javadi, and R. Buyya, *Resource provisioning based on leases preemption in InterGrid*. In Proceedings of the 34th Australasian Computer Science Conference (ACSC'11), pages 25–34, Perth, Australia, 2011.
- Chapter 4 deals with the proactive contention-aware scheduling in the InterGrid Gateway (IGG) level. This work sought to avoid the resource contention within a Grid by proactive scheduling of external requests on resource providers. Chapter 4 derived from:
  - **M. Amini Salehi**, B. Javadi, and R. Buyya, *QoS and preemption-aware scheduling in federated and virtualised grid computing environ-*

ments. Journal of Parallel and Distributed Computing (JPDC), 72(2):231–245, 2012.

  - **M. Amini 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), pages 419–432, Australia, 2011.
  - **M. Amini Salehi** and R. Buyya, *Contention-aware resource management system in a virtualised grid federation*, in PhD Symposium of the 18th international conference on High performance computing (HiPC'11), India, 2011.

- Chapter 5 concentrates on the long response time for external requests in the resource providers and proposes a contention-aware admission control policy to prevent long response time for external requests. Chapter 5 derived from:
  - **M. Amini Salehi**, B. Javadi, and R. Buyya, *Preemption-aware admission control in a virtualised grid federation*. In Proceedings of 26th International Conference on Advanced Information Networking and Applications (AINA'12), pages 854–861, Japan, 2012.
- Chapter 6 considers the impact of energy-efficient mechanisms on the waiting time of external requests in InterGrid and reduces the energy consumption within an RP without affecting the required performance of the system. Chapter 6 derived from:
  - **M. Amini Salehi**, P. R. Krishna, K. S. Deepak, and R. Buyya, *Preemption-aware energy management in virtualised datacenters*, in Proceedings of 5th International Conference on Cloud Computing (IEEE Cloud'12), USA, 2012.
- Chapter 7 describes the realisation of resolving contention in InterGrid. We detail the system implementation and evaluate the performance of different preemption policies to reduce the side-effects of applying preemption mechanism. Chapter 7 derived from:
  - **M. Amini Salehi**, A. N. Toosi, and R. Buyya, *Realising Contention-aware Scheduling in InterGrid*, submitted to Software: Practice and Experience (SPE) journal, 2012.
- In Chapter 8, we present general considerations, conclusions, and future directions.

Chapter 2

Principles, Taxonomy, and Context

This chapter discusses approaches for contention management in resource sharing distributed computing systems. The chapter investigates features of these approaches, identifies, and categorises their similarities and differences. Key background information required for better understanding of the topics discussed in the remaining chapters, in addition to the positioning of this dissertation in regards to related works, are also provided.

2.1 Introduction

During our daily life, we commonly use several large-scale distributed systems: Communicating through telephone lines, utilising electricity for lights, using airline companies that work together to reach us to a destination, and etc [18]. These distributed systems usually serve clients with different priorities. For instance, an airline company partitions seats to different classes for various clients. Telephone companies, as another instance, secure bandwidth for governmental institutions. Distributed computing systems, such as Grids and Clouds, are another type of distributed systems that support users with distinct priorities.

A distributed computing system is essentially a set of computers that share their resources via a computer network and interact with each other towards achieving a common goal [19]. The shared resources in a distributed system vary and can 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 a scalable Internet application.

Distributed systems such as Clusters, Grids, and recently Clouds have be-

come ubiquitous platforms for supporting resource-intensive and scalable applications. However, surge in demand is still a common problem in distributed systems [1] in a way that no single system (specially Clusters and Grids) can meet the needs of all users. This has led to emergence of resource sharing between distributed systems and the notion of interconnected distributed systems.

In an interconnected distributed system, as depicted in Figure 2.1, organisations (referred as resource providers in this dissertation) share their resources over the Internet and consequently are able to access larger resources. In fact, interconnected distributed systems construct an overlay network on top of the Internet to facilitate the resource sharing between the constituents.

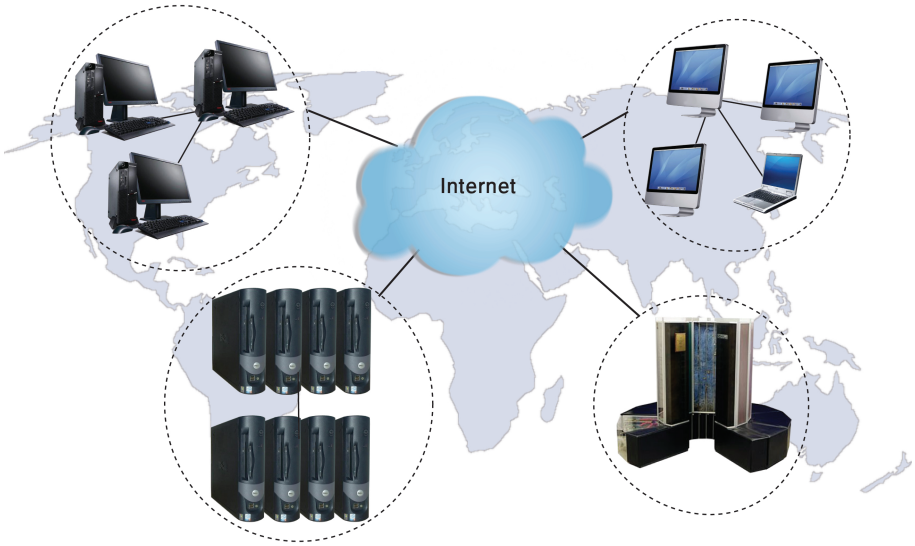


Figure 2.1: Interconnected distributed systems.

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 categorised in Figure 2.2 and explained over the following paragraphs.

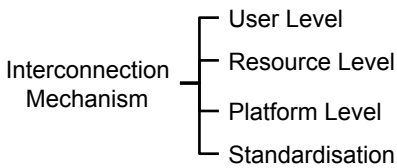


Figure 2.2: Interconnection mechanisms in distributed systems.

- User level (Broker-based): In this approach, an end-user employs a software tool [20, 21] that is connected to multiple distributed systems and creates a loosely coupled interconnection between them. This approach involves repetitive efforts to develop interfaces for different distributed systems thus, it is difficult to scale to many distributed systems. Gridway [20] and GridBus broker [21] are examples of broker-based interconnection approach. The former achieves interconnection in organisation level, whereas the latter, works in the end-user level.

- Resource level: In this approach, interfaces to different middlewares are developed on the resource provider side, consequently the resources can be available to multiple distributed systems. Difficulties of developing interface for different middlewares is an obstacle to scale to many distributed systems. Additionally, system administrator of the provider has to have the knowledge of dealing with the different middlewares. Hence, this approach is suggested for large distributed systems. Interconnection of EGEE, NorduGrid, and D-Grid is carried out based on this approach [19]. Particularly, D-Grid [22] achieves interconnectivity via implementing interfaces for UNICORE, gLite, and Globus on each provider in a way that resources can be accessed by any of the middlewares.

- Platform level (Gateway): 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 points of failure and also a scalability bottleneck. InterGrid [1] and the interconnection of Naregi and EGEE [23] are instances of this approach.

- Common interfaces: Common and standard interfaces have been accepted as a comprehensive and sustainable solution for interconnection of distributed systems [19]. 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 [24] is a large-scale interconnected distributed system implemented based on a proposed 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 [25], Condor [26], and Ganglia [27] 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 [28].

Grid computing is a prominent example of interconnected distributed systems. Although Grids include different types of resources, this dissertation focuses on computational resources, thus a *Grid is considered as a collection of autonomous resource providers (e.g., Clusters) that have their own Local Resource Management Systems* [2]. Nowadays, Grids are utilised predominantly in scientific communities to run high performance computing (HPC) applications [29–31]. Over the last decade, a variety of Grids have emerged based on different interconnection mechanisms. TeraGrid in the US [32], DAS in the Netherlands [33], and Grid5000 in France [34] are such examples.

Generally, in an interconnected environment, such as Grids, requests from different sources and with different priorities co-exist. 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, accordingly, there are different ways to cope with these contentions.

In addition to Grids, resource contention can also occur in compute Clusters and Clouds. In this dissertation, we define a compute Cluster as *a collection of connected computers that work together and is managed by a resource management system*, such as Maui [35] and OpenNebula [36]. Also compute Clouds (Infrastructure as a Service) in this dissertation are considered as *large-scale distributed systems where computational resources are provided in form of virtual machines as demanded by customers* [37].

This chapter concentrates on the field of resource contention within interconnected distributed systems, particularly Cluster, Grid, and Cloud platforms. It summarises the key concepts and provides an overview of the most prominent approaches.

2.2 Request Management in Interconnected Distributed Systems

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

- Scientists in a research organisation 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). Nowadays, offering a combination of advance-reservation and best-effort schemes [13], interactive and batch jobs [38], and tight-deadline and loose-deadline jobs [39] are common practices in interconnected distributed systems.

Requests with diverse QoS levels contend to access resources specially when there is a surge in demand. This contention is challenging and should be handled by resource management systems. Specifically, *resource contention* occurs when a user request cannot be admitted or cannot receive adequate resources because the resources are occupied by other (possibly higher priority) requests.

There are different approaches to resolve resource contention in resource management systems level. One common approach is prioritisation of requests based on criteria such as Quality of Service (QoS) or origin. For instance, in an interconnected distributed system local requests (i.e., local organisations'users) typically have priority over requests from external users [40]. Another example is in urgent computing [41] (urgent applications), such as earthquake and bush-fire prediction applications, where the applications intend to acquire many resources in an urgent manner.

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

2.3 Resource Contention in Interconnected Distributed Systems

Contention to access resources frequently occurs in different systems. For example, customers contend to buy tickets from an airline company. In this situation, the company categorises (partitions) the seats for different customers. Another example of contention for resources is when a disaster, such as a bushfire, occurs in a place. In this situation, resources (e.g., machineries, human resources) can be preempted from their normal task to recover from the disaster.

In this section, we concentrate on the causes of resource contention in distributed computing systems. Additionally, we investigate possible solutions for different types of contentions. Figure 2.3 demonstrates the taxonomy of different contention types along with their solutions.

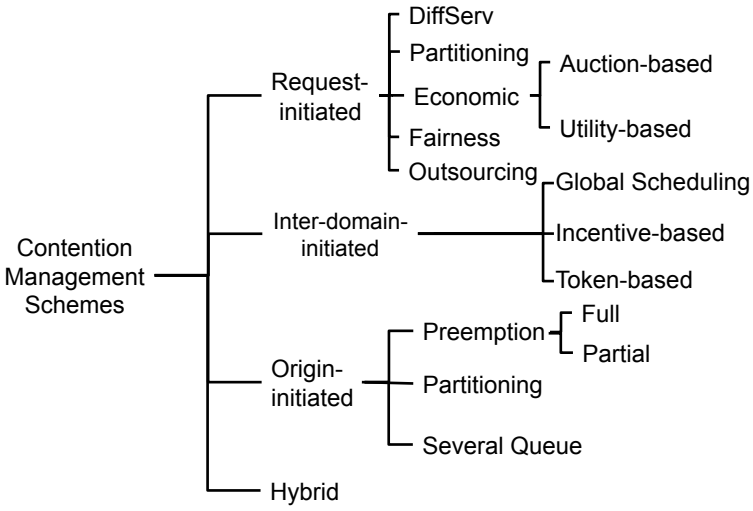


Figure 2.3: Taxonomy of different types of resource contentions and possible solutions in distributed systems.

2.3.1 Request-initiated Resource Contention

This type of resource contention occurs when a request monopolises resources to such an extent that deprives others from accessing them. Request-initiated contention is prevalent in all forms of distributed systems. Request-initiated contention takes place for 2 main reasons.

The first reason is imbalance in size of requests, particularly in terms of number of nodes required or their execution time (duration) [42]. In this circumstance, small requests may have to wait for a long time behind a long one to access resources.

The second reason for request-initiated contention is 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:

- Hard QoS: Where the QoS constraints cannot be negotiated. These systems are prone to QoS violation hence, managing resource contention is critical [42].
- 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 [42].

- Hybrid QoS: Where the resource management system supports a combination of Hard and Soft QoS requirements for user requests. This manner 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. Resource management systems that support combination of interactive (hard QoS) and batch requests (usually soft QoS) [38] are other instances of supporting Hybrid QoS requirements.

Approaches for managing request-initiated contention are mostly applied in the context of scheduling and/or admission control units of resource management systems. Over the next paragraphs, we categorise and describe different approaches for handling this type of resource contention.

The *Differentiated Services* (DiffServ) approach was initially 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 division of services into distinct QoS levels. According to IETF RFC 2474, each level is supported by dropping TCP packets of lower priority levels.

Similar approach is applied in the context of request-initiated resource contention 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, the DiffServ approach is appropriate for soft QoS requirements.

Variations of the DiffServ approach can be applied when contention occurs due to imbalance in request characteristics. For example, Silberstein et al. [43] sought to decrease the response time of short requests in a multi-grid environment. For that purpose, they apply a multi-level feedback queue (MLFQ) scheduling policy. 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 be completed in the time limit of that level, they are migrated to the lower level queue which is a larger grid. The process continues until the task finishes or reaches the bottom of the hierarchy.

In the *Partitioning* approach, the resources are reserved for requests with

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

The *Economic* approach that works either in 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 maximise the utility based on the valuation function and makes available 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 maximises the utility function has the priority to access resources. These approaches are commonly applied in market-oriented scheduling [44].

The *Fair Scheme* guarantees that contentious requests receive their share of the system resources [45]. This approach is used to resolve resource contentions resulting from imbalanced requests in the system and assures an starvation-free scheduling for the requests.

The *Outsourcing* approach is applicable in the interconnected distributed systems. In fact, interconnection of distributed systems creates the opportunity to employ resources from other distributed systems in the case of resource contention. Outsourcing approach is applied for both causes of request-initiated resource contention (i.e., request imbalance and QoS levels). Specially, Cloud providers are extensively employed for outsourcing requests [46]. This issue has helped in the emergence of hybrid Clouds, which are a combination of private (organisational) resources and public Clouds [47]. Although we categorise outsourcing as a resolution for request-initiated contentions, it can be applied for inter-domain and origin initiated contentions as discussed next.

2.3.2 Inter-domain-initiated Resource Contention

Inter-domain-initiated resource contention occurs in interconnected distributed systems, when the proportion of shared resources to the consumed resources by a constituent distributed system is low. In other words, this type of resource contention happens when a resource provider contributes few resources while demands a lot of resources from other resource providers. Unlike request-initiated contention, which merely roots in request's characteristics and can take place in any distributed system, inter-domain contention is based on the overall consumption and contribution of each resource provider.

There are several approaches for handling inter-domain-initiated contention, including global scheduling, incentive, and token-based approaches (see Figure 2.3).

Global schedulers: In this approach 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 [26] or Sun Grid Engine (SGE) [48], allocate resources to the requests. A global scheduler handles inter-domain contention by admitting requests from various organisations based on the number of requests it has redirected to them. Since global schedulers usually are not aware of the instantaneous load variations in the resource providers, it is difficult for them to guarantee QoS requirements of users [3].

Incentive approach: In this approach, which is mostly used in peer-to-peer systems [49], resource providers are encouraged to share resources to be able to access more resources. Reputation Index Scheme [50] is a type of incentive-based approach in which the organisation cannot submit requests to another organisation while it has less reputation than that organisation. Therefore, in order to gain reputation, organisations are motivated to contribute more resources to resource sharing environment.

Quality-service incentive scheme [51] 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 approach: Where resource contention is resolved based on the number of tokens allocated to resource providers. The number of tokens for each provider is proportional to its resource contribution. If a user wants to get access to another organisation resources, her consumer agent must spend a certain amount of tokens to get the access.

This approach can resolve request-initiated contention as well as inter-domain contentions. To resolve the request-initiated resource contention, valuation functions can be used to translate the QoS demands of a user to the number of tokens should be used for a request. The provider agent can use its own valuation functions to compute the admission price for the request. The request will be admitted only if the admission price is less or equal to the number of tokens that the requesting organisation is willing to spend [42].

2.3.3 Origin-initiated Resource Contention

In interconnected distributed systems, users'requests originate from distinct organisations. Therefore, these systems are prone to resource contention between

local requests of the organisation and requests arriving from other organisations (i.e., external requests). Typically, local requests of each organisation have priority over external requests [40]. In other words, the organisation 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 inter-domain-initiated and request-initiated resource contentions. Consequently, the approaches of tackling this type of resource contention are similar to the already mentioned approaches. Particularly, partitioning approach both in static and dynamic forms and global scheduling are applicable to resolve origin-initiated resource contentions. There are also other approaches to cope with origin-initiated contentions that are discussed in this section.

Preemption: This approach interrupts the execution of a request and frees resources for another, possibly of 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 checkpointing is not supported in a system, then the preempted request can be killed (canceled) or restarted. For parallel requests, usually *full preemption* is performed, in which the whole request leaves the resources. However, some systems support *partial preemption*, in which part of resources allocated to a parallel request is preempted [52].

Although preemption mechanism is a common solution for origin-initiated contention, it is also widely applied to solve request-initiated resource contention. Due to the prominent role of preemption in resolving different types of resource contentions, in Section 2.5 we explain it in details.

Partitioning: Both static and dynamic partitioning of resources, as mentioned in Section 2.3.1, can be applied to tackle origin-initiated contention.

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 [3].

Several Queues: In this approach when requests arrive [4], they are categorised in distinct queues, based on their origin. Each queue can have its own scheduling policy. Then, another policy determines the appropriate queue that can dispatch a request to resources.

Combinations of the aforementioned contentions (mentioned as hybrid in

Figure 2.3) can occur in an interconnected distributed system. Particularly, the combination of origin-initiated and request-initiated resource contention commonly occurs in interconnected distributed systems. 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 [16, 53, 54]. Generally, resolution of hybrid resource contentions is a combination of different strategies mentioned above.

2.4 Contention Management in Resource Management Systems

Resource management system in a distributed system is responsible for resolving resource contention. In this section, we explore how architectural elements of a resource management system can contribute in resolving various types of resource contention. Figure 2.4 summarises this section by presenting different components of a resource management system and the way they handle resource contention.

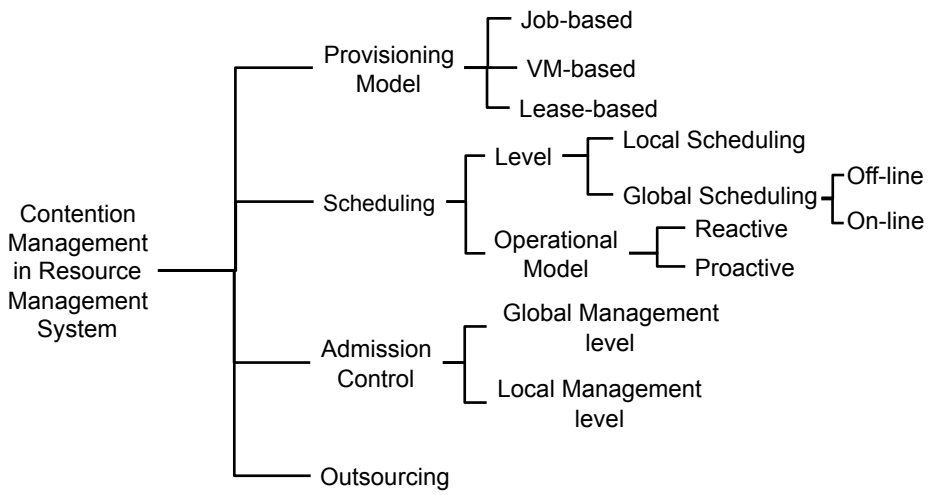


Figure 2.4: Components of a resource management system that can help in resolving resource contention in interconnected distributed systems.

2.4.1 Resource Provisioning

The resource provisioning component of a resource management system is in charge of procuring resources based on user application requirements. Resource provisioning in a system is performed based on a provisioning model that defines the execution unit in that system. Resource provisioning models do not directly

deal with resource contention. However, the way other components of resource management system function depends on the resource provisioning model.

Provisioning resources for users'requests in distributed systems has 3 dimensions as follows:

- Hardware resources.
- Software available on the resources.
- Time during which the resources are available (availability).

Satisfying all of these dimensions in a resource provisioning model has proved to be challenging. Past resource provisioning models in distributed systems were unable to fulfil all of these dimensions [13]. Emergence of virtual machine (VM) technology as a resource provisioning model recently has created an opportunity to address these dimensions. Over the next subsections, we discuss common resource provisioning models in current distributed systems.

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 of a job is the consequence of a request to run the job. Job model resource provisioning has been widely employed in distributed systems. However, this model cannot perfectly support all resource contention solutions [2].

Job-based systems provision hardware resources for jobs while they offer a limited support for software availability. In fact, in job-based model users do not have administrative access to resources, 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. [55], presented advance-reservation model over the job-based provisioning model. They support advance-reservation through prediction of waiting time of jobs in the queue. Hovestadt et al. [56], enabled advance-reservation through plan-based scheduling that finds the place of each job to instead of waiting in the queue. In this system, upon arrival of each job, the whole schedule is re-planned to optimise the resource utilisation.

Falkon [57], Condor glidin [58], MyCluster [59], and Virtual Workspace [60] apply a multi-level/hierarchical scheduling on top of a job-based system to of-

fer other provisioning models (such as the lease-based model described in Section 2.4.1). In these systems, one scheduler allocates resources to another scheduler and the second scheduler runs the jobs on the allocated resources [61].

Virtual Machine Model

Virtual Machines (VMs) are considered an ideal vehicle for resource provisioning in distributed systems. In VM model, hardware, software, and availability can be provisioned for user requests. Additionally, VMs'capability in getting suspended, resumed, or migrated without major utilisation loss has proved to be useful in resource management. Therefore, VM-based provisioning model is extensively used in current distributed systems.

The VM-based resource provisioning model is used for creation of Virtual Clusters on top of an existing infrastructure. Virtual Clusters (VC) are usually utilised for job-based batch processing. For example, in MOSIX [62], Clusters of VMs are transparently created to run high performance computing (HPC) applications. The Nimbus toolkit [63] provides "one-click Virtual Cluster" automatically on heterogeneous sites through contextualised disk images. Amazon EC2 provides VM-based Cluster instances[1] that offer supercomputer services to expedite the execution of HPC applications. Automatic VM creation and configuration is also considered in In-VIGO [64] and VMplants [65]. An extension of Moab [66] creates VM-based Virtual Clusters to run HPC batch applications.

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

Datacenters usually contain large-scale computing and storage resources (order of 100s to 1000s) and consume a significant amount of energy. A remarkable benefit of deploying VM-based provisioning model in datacenters is the consolidation feature of VMs that can potentially reduce energy consumption [67]. However, VM consolidation requires accurate workload prediction in the datacenters. Moreover, the impact of consolidation 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 are suspended and resumed on another resource, which involves time overhead. In the latter approach, live migration [68] of VMs is used, thus, it is transparent for the user.

Solutions such as VMware, Orchestrator, Enomalism, and OpenNebula [36]

[1] http://aws.amazon.com/hpc-applications/

provide resource management for VM-based data centres.

There are also concerns in the deployment of VM-based provisioning model and Virtual Clusters. Networking and load balancing amongst physical Clusters is one of the challenges that is considered in Vio-Cluster [9]. Power efficiency aspect and effectively utilisation of VMs capability during suspension and migration are also considered by many researchers [69–71]. Overhead and performance issues involved in utilisation of VMs to run compute-intensive and IO-intensive jobs, fault tolerance, and security aspects of VMs are also of special importance in the deployment of VM-based provisioning model.

Lease Model

This model is considered as an abstraction for utility computing in which the user is granted a set of resources for a specific interval and with an agreed quality of service [72]. In this model job execution is independent from resource allocation, whereas in the job model resource allocation is the consequence of a job execution.

Formally, a *lease* is defined by Sotomayor [13] 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 the resource management system, then users are 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 for implementing lease-based model. Depending on the contract and the features of resource management system, resource procurement for leases can be achieved from a single provider or multiple providers.

2.4.2 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 increase the amount of resource contention.

In an interconnected distributed system, we can recognise 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). In the next step, the local scheduler performs further tuning to run the assigned requests efficiently on re-

sources.

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

Local Scheduling

Local scheduler is the main component of the local resource management system (LRMS) in each resource provider. Scheduling policies in this level mainly deal with request-initiated and origin-initiated contentions. There are few local schedulers that handle inter-domain-initiated contention in this level.

Backfilling is a common scheduling strategy applied in LRMS. The aims of backfilling is to reduce queuing fragmentation, as a result it increases resource utilisation and minimises the average response time of requests. In fact, backfilling is an improved version of FCFS policy in which requests that arrive later, possibly are allocated earlier in the queue, if there is enough space for them. Variations of the 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. [10] applied preemption on the 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 the resource utilisation increases. The preempted request is restarted and rescheduled in the next available time-slot.

Multiple resource partitioning is another scheduling approach for local schedulers proposed by Lawson et al. [4]. In this approach, resources are divided into partitions that potentially can borrow resources from each other. Each partition has its own scheduling policy. 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 the FCFS and 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. Advance-reservation is supported in resource management systems such as LSF [73], PBSPRO [74], and Maui [35].

Advance-reservation is prone to low resource utilisation specially if the reserved resources are not used. Additionally, it increases the response time of normal requests [75, 76]. These side-effects of advance-reservation can be minimised by limiting the number of advance-reservation, and leveraging flexible advance-reservation (in terms of start time, duration, or number of processing elements needed).

Global Scheduling (Meta-scheduling)

Global scheduler in an interconnected distributed system, such as InterGrid, usually has two aspects. On the one hand, the scheduler is in charge of assigning incoming requests to RPs (e.g., Clusters) within its domain. On the other hand, it is responsible to deal with other distributed systems through schedulers or gateways that delegate them. These aspects of global schedulers are useful in resolving inter-domain-initiated and origin-initiated resource contention.

Global schedulers in interconnected distributed systems are the entry points to each constituent distributed system. Accounting information regarding requests that are sent or received to/from other distributed systems can be used to resolve the inter-domain contentions in these schedulers. Global schedulers can also proactively schedule requests within their domain in a way that origin-initiated resource contention is avoided. For that purpose, the scheduler has to consider the likelihood of contention in each provider based on the local workload condition in each provider.

The global scheduler either works off-line (i.e., batches incoming requests and assigns each batch to an LRMS), or on-line (i.e., assign each request to an LRMS upon arrival).

2.4.3 Admission Control Unit

Controlling the admission of requests prevents imbalanced resource deployment. By employing an appropriate admission control policy, various types of resource

contentions can be either avoided or handled. 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 behaviour should depend on the workload condition in an RP. Applying a strict admission control in a lightly loaded system results in low resource utilisation and high rejection of requests. Nonetheless, the consequence of applying less strict admission control in a heavily loaded resource is more QoS violation and user dissatisfaction [77].

One common way to tackle request-initiated contention in admission control is introducing a valuation function [42]. The function relates the quality constrains of users to a single quantitative value. The value indicates the amount a user is willing to spend for a given QoS. Resource management systems use the valuation functions to allocate resources with the aim of maximising aggregate valuation for all users.

Admission control can be applied to resolve inter-domain-initiated contention by limiting the amount of accepted requests of each organisation proportional to their resource contribution. Admission control can be applied to avoid origin-initiated resource contention. For this purpose, an admission control policy would accept external requests that their QoS constraints can be fulfilled based on the workload of local requests.

In an interconnected distributed system, admission control can be performed in the LRMS and/or along with the global scheduler. In the former, for rejecting a request there should be an alternative policy to manage the rejected request. For instance, the rejected request can be redirected to another resource provider or even queued in a separate queue and scheduled in a later time. Admission control in the global scheduler level can reject an external request by notifying the requester peer scheduler. However, the drawback of employing admission control with global scheduler is that the global scheduler may not have updated information about the workload situation in the resource providers.

2.4.4 Outsourcing Unit

Interconnectivity of distributed systems creates the opportunity to resolve resource contention via deployment of resources from other distributed systems. Therefore, resource management systems in interconnected distributed systems usually have a unit that decides about details of outsourcing requests (i.e., redi-

recting 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 admission control, scheduling, or both of these units [47].

Outsourcing is generally applied when there is a peak demand or there is a resource contention (specially request-initiated contention) [78]. In this situation, to serve requests without contention, some of them (e.g., requests with long waiting time) are selected to be redirected to other distributed systems. Cloud computing providers are of special interest to be employed for outsourcing (off-loading) requests [46].

2.5 Preemption Mechanism

Preemption mechanism in a resource management system vacates resources 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 the preemption mechanism is implemented depends on the way the checkpointing operation is performed. If the checkpointing is not supported, then the preempted process has to be cancelled and restarted at a later time. If checkpointing is supported both in the process level and in the scheduler levels, 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 2.5.4.

Due to the critical role of preemption mechanism in as an approach to resolve resource contention, in this section, we investigate preemption in distributed systems from various angles. Particularly, we consider various usages of preemption mechanism and the way they resolve resource contention. Then, we investigate possible side-effects of preemption mechanism. Finally, we discuss how a preempted request (which can be in form of a job, VM, or a lease) can be resumed in a distributed system.

2.5.1 Applications of Preemption Mechanism

In this part, we investigate the preemption in distributed systems and identify how different types of resource contentions can be resolved using preemption.

Additionally, we discuss other usages of preemption in distributed systems. The taxonomy of preemption usages is presented in Figure 2.5.

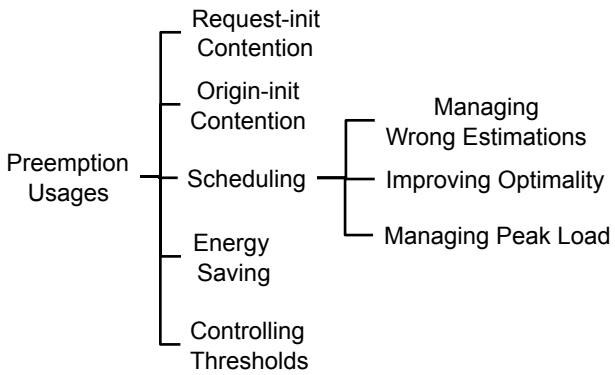


Figure 2.5: Different usages of preemption mechanism in distributed systems.

Preemption mechanism is used to resolve request-initiated resource contentions. One approach is to employ preemption along with the local scheduling policy (e.g., backfilling) to prevent unfairness in the scheduling [79]. For instance, when a backfilled request exceeds the allocated time-slot and interferes with the reservation of other requests, the preemption mechanism can suspend the backfilled requests to start scheduled reservations on time [12, 80]. The preempted request can be allocated another time-slot to finish its computation at a later time.

MOSIX is a job-based platform for high performance computing on Linux-based Clusters. A preemptive scheduling algorithm is implemented in MOSIX [80] to allocate excess resources to jobs that require more resources than their share. However, these resources are released as soon as they are reclaimed.

Scojo-PECT [81] provides a limited response time for several job classes within a virtualised Cluster. To cope with the request-initiated resource contention, it employs the DiffServ approach that is implemented via coarse-grained preemption. 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. [38] introduced a preemption-based scheduling policy for batch and interactive jobs within a virtualised Cluster. In this work, batch jobs are preempted in favour of interactive jobs. The authors introduce various 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, including the time requests spent in the queue.

Haizea [13], is a lease scheduler that schedules a combination of advance-reservation and best-effort leases. Haizea preempts best-effort leases in favour of advance-reservation requests. Haizea also considers the overhead time imposed by preempting a lease, including the time for suspending and resuming VMs within a Cluster.

Preemption of parallel jobs has been implemented in the Catalina job scheduler [75] in San-Diego Supercomputer Center (SDSC) by adding preemption to the conservative backfilling scheduling policy. 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, preemption of jobs with larger size (wide jobs) implies more overhead because of the time needed for saving messages between nodes.

Isard et al. [82] investigated the problem of optimal scheduling for data-intensive applications, such as Map-Reduce, on 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 an arriving job.

Preemption can be applied to resolve origin-initiated resource contention. Ren et al. [8], proposed a prediction method for unavailable periods in fine-grained cycle sharing systems where there is mixture of local jobs and global (guest) jobs. The prediction is used to allocate global requests in a way that they do not delay local requests.

Gong et al. [7] have considered preemption of external tasks in favour 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 that is distributed over a NOW.

MOSIX resolves origin-initiated contention between local and guest (external) jobs by providing preemptive priority for the local jobs and migrating preempted external jobs [40].

Apart from resolving resource contention, preemption mechanism has other usages such as improving the quality of scheduling policies. Preemption mechanism can be used as a tool by scheduler to enforce its decisions.

Scheduling algorithms in distributed systems are highly dependent on the user estimations of the requests'runtime. There are studies (e.g., [83]) that demonstrate the inefficiency of these estimations and how these wrong estimation can compromise the scheduling performance. In the presence of inaccurate estimations, preemption mechanism can be used 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 advance-reservation. In this situation preemption mechanism abstracts the scheduling policy from the obstacles in enforcing that policy [12].

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 [80]. Therefore, preemption can potentially mitigate the non-optimality 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 [15].

Preemption mechanism can be employed to improve the system and/or user centric criteria, such as resource utilisation and average response time. Kettimuthu et al. [84] investigated the impact of preemption of parallel jobs in supercomputers for improving the average and worst case slowdown of jobs. They propose 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 mechanism is in energy conservation of datacenters. One prominent approach in energy conservation of virtualised datacenters is VM consolidation, which takes place when resources in the datacenter are not utilised efficiently. In VM consolidation, VMs running on under-utilised resources are preempted (suspended) and resumed on other resources. VM consolidation can also occur through live migration of VMs [68] to minimise the unavailability time of the VMs. When a resource is evacuated, it can be powered off to reduce the energy consumption of the datacenter.

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 the system to automatically preempt part of the load and reschedule on other available nodes. Bright Cluster Manager [85] is a commercial Cluster resource management

system that offers the ability to establish preemption rules based on metrics and thresholds.

2.5.2 Challenges of Preemption Mechanism

Operating systems of single processor computers have been applying preemption mechanism for a long time to offer interactivity to end-users. 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. These challenges are different based on the resource provisioning model.

In this part, we present details of challenges that distributed systems encounter in preempting requests based on their resource provisioning models. Moreover, a summary of preemption challenges based on different provisioning models is provided in Table 2.1.

- **Coordination:** Distributed requests are scattered on several nodes by nature. Preemption of distributed requests has to be coordinated between nodes that execute them, regardless of the type resource provisioning model is used. 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. Since VM- and lease-based systems are self-contained (isolated) by nature, there is not usually security concern in their preemption.

- **Checkpointing:** The absense 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 cancelled, which is a waste of resources [10]. Checkpointing problem is obviated in VM- and lease-based resource provisioning models [17]. Due to the fundamental role of checkpointing in the preemption mechanism, in Section 2.5.4 we discuss it in details.

- **Time overhead:** In VM- and lease-based resource provisioning models, the time overhead imposed to the system to perform preemption is a major challenge. If preemption takes place frequently and the time overhead is not negligible, then the resource utilisation will be affected.

Additionally, ignoring preemption time overhead in scheduling prevents requests to start at the scheduled time [17]. 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 delays the start of leases, which subsequently might violate QoS requirements.

Sotomayor et al. [17] have presented a model to predict the preemption time overhead for VMs in a Cluster. 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 in the system has to be reduced [86].

- **Permission:** In the lease-based resource provisioning model, preemption of 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 notification of user (requester), whereas leases require the requester's permission for preemption [72]. 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.

Table 2.1: Challenges of preemption mechanism in different resource provisioning models.

	Resource Provisioning Model		
Challenge	**Job-based**	**VM-based**	**Lease-based**
Coordination	✓	✓	✓
Security	✓	✗	✗
Checkpointing	✓	✗	✗
Time overhead	✓	✓	✓
Permission	✗	✗	✓
Impact on queue	✓	✓	✓
Long response time	✓	✓	✓
Preemption candidates	✓	✓	✓

- **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. Preemption of the running processes and allocation of 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.

- **Long response time:** Preemption leads to increasing the waiting time for low priority requests [84]. 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 the instablity and long response time. One approach to cope with the long response time challenge is restricting the number of requests admitted in a distributed system. A preemption policy was presented by Walter et al. [38] in a VM-based system with the objective of avoiding long response time for batch requests where a combination of batch and interactive requests co-exist in the system.

- **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 high priority requests. Therefore, there are several sets of candidate requests whose preemption can create adequate space for high priority requests. As it is expressed in Figure 2.6, there are several candidate sets (Figure 2.6(b)) whose preemption can vacate resources for the required time interval (i.e., from t1 to t2 as indicated in Figure 2.6(a)).

  Selecting distinct candidate sets affects the amount of unused space (also termed scheduling fragment) in the schedule. Furthermore, preemption of 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 challenging. To cope with this challenge, the backfilling policy has been extended with preemption ability in the Maui scheduler [10] to utilise scheduling fragments.

2.5.3 Possibilities for Preempted Requests

Issues discussed thus far are related to the preemption mechanism and its challenges. However, making a proper decision for the preempted request is also important. This decision depends on the administrative policies of resource providers as well as their resource provisioning model. For example, migration may not be possible in a particular job-based distributed system.

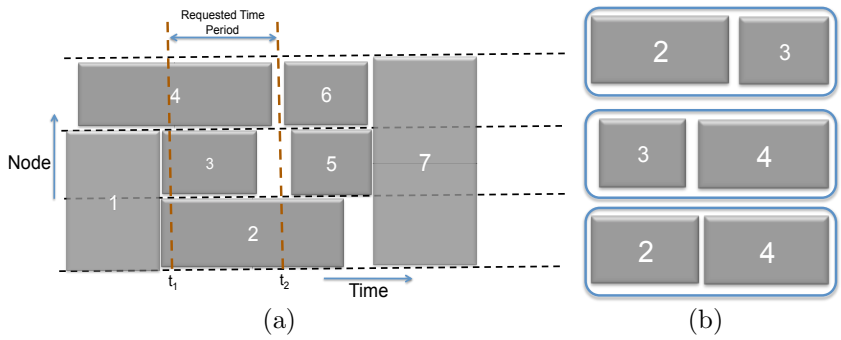


Figure 2.6: Preemption candidates for a request for two nodes. Figure 2.6(a) shows collision of the requested time interval with running requests within a scheduling queue. Figure 2.6(b) presents different candidate sets whose preemption creates space for the new request.

Thanks to the flexibility offered by 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 cancelation or suspension and resuming of the preempted job. Focusing on the VM-based resource provisioning model, we introduce various cases that can possibly happen for preempted requests. Additionally, Figure 2.7 expresses the VM's life-cycle based on different cases for VM preemption.

- **Cancelation:** VMs can be canceled (terminated) with or without notification of 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 are an example of cancelation of VMs. 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 jobs is a common practice [10] because of the difficulty of performing other possible actions.

- **Restarting:** In both job-based and VM-based systems, the preempted request can be cancelled and restarted either on the same resource or on another resource. The disadvantage of this choice is the loss of preliminary results and wasting of 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.

- **Malleability (partial preemption):** In this approach, the number of nodes/VMs allocated to a request might vary while it is executing. The

running request should be able to tolerate such variations. Specifically, this operation can be performed on malleable jobs [87] in job-based systems. In VM and lease-based systems, frameworks such as Cluster-on-Demand (COD) [11], support this manner of preemption via cancelation of a fraction of VMs of a request. Malleability is also known as partial-preemption and can be used to implement dynamic partitioning (see Section 2.3.1).

- **Pausing:** When a VM is paused, it does not get any CPU share, however, it remains in the host memory. Resumption of the VM, in this case, happens by getting the CPU share and is very fast (Figure 2.7). In fact, pausing is not considered as a complete preemption operation. Nonetheless, the main usage of pausing is to perform lease-level preemption. When a lease is preempted, in order to prevent inconsistency or message loss, all VMs are paused and then, the suspension takes place [12]. This is shown in Figure 2.7 as a link between pause state and sleep (suspended) state. More details on the usage of pausing for preempting leases are discussed in Section 2.5.4.

- **Suspending:** When a VM is suspended, its state, including the memory contents as well as the state of all processes running within the VM, is saved to the disk. The suspended request has to be rescheduled to find another free time-slot for the remainder of its execution. At resumption time, the disk image is re-loaded into the memory and the VM resumes from the suspended point. 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 important particularly for data-intensive applications. A suspended request can be resumed in one of the 3 following ways:

  - Resuming on the same resource; This case does not yield to optimal utilisation of whole resources.
  - Resuming on the same resource provider but not essentially on the same resource; In this case, usually data transfer is not required.
  - Resuming on different resource provider: This case leads to migration to another provider and entails data transfer operation; This is particularly not recommended for data-intensive applications.

  Based on the Figure 2.7, suspension VMs can be performed directly from the running state. However, as mentioned earlier, to suspend a lease that includes several VMs, pause operation should be performed before the suspension to assure the consistency of the jobs running inside the lease.

- **Migration:** VMs of the preempted request are moved to another resource provider to resume the computation (also called cold migration). According to Figure 2.7, migration involves suspending the VM on the disk, transferring its disk image, and resuming VMs in the destination resource provider. Transferring the disk image over the network is the major overhead in the migrating operation [88]. One solution to mitigate this overhead in interconnected distributed systems is migrating to a resource 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 [13].

- **Live-Migration:** As shown in Figure 2.7, with live migration, preemption can be carried out without major interruption in running VMs. This is particularly essential in conditions that interruption cannot be tolerated, such as Internet servers. Current techniques for live migration just transfer the dirty pages of VMs to decrease the overhead.

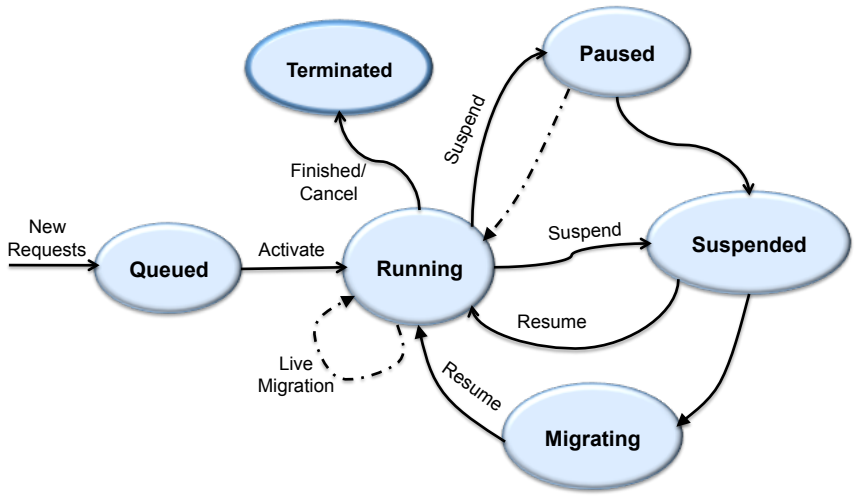


Figure 2.7: VM life-cycle by considering different possible preemption decisions in a resource management system.

Apart from the above choices, there are requests that cannot be preempted (i.e., non-preemptable requests). For example, critical tasks in workflows that have to be started and finished at exact times to prevent delaying the execution of the workflow [89]. Another example is secure applications that cannot be moved to any other provider and cannot also be interrupted in the middle of their execution.

In a particular resource management system, one or combination of the mentioned operations can be performed on the preempted request. The performed

action can be based on QoS constraints of the requests or restrictions that user declares for the request. Another possibility is that the resource management system dynamically decides the appropriate operation to be performed on the preempted request.

2.5.4 Checkpointing in Distributed Systems

Checkpointing is the function of storing the latest state of a running process and is required in all types of resource provisioning models. 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 requests.

A 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 2.8. In this section, we explain checkpointing strategies for different provisioning models in distributed systems.

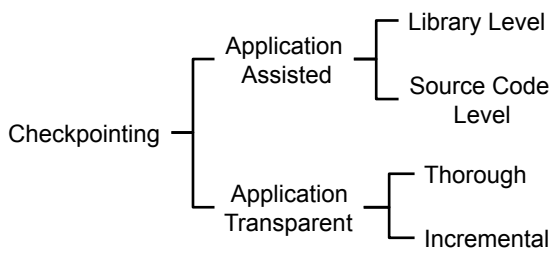


Figure 2.8: Different checkpointing methods in distributed systems.

Checkpointing in Job-based Provisioning Model

According to Figure 2.8 checkpointing approaches are categorised as application-transparent and application-assisted. In application-assisted (user-level) checkpointing, the application defines the necessary information (also called critical data area) that has to be checkpointed. The disadvantage of this approach is that it entails modification of 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 categorised as follows:

- Source-code level: In this manner, checkpointing codes are hard-coded by developers. There are also some source code analysis tools [90, 91] that help developers to find out the suitable places where checkpointing codes can be inserted.

- Library level: There are libraries for checkpointing, such as Libckpt [92] and Condor libraries [93]. To use this kind of checkpointing, developers have to recompile the source code by including the checkpointing library in their program.

As expressed in Figure 2.8, checkpointing can also be performed 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 and it is not needed to be modified to be checkpointable. Application-transparent checkpointing technique is particularly applied in preemption whereas application-assisted scheme is more used for fault-tolerance purposes. Examples of system level checkpointing include BLCR [94] and CRAK [95].

Since 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 dependency on a specific version of the operating system that they are operating on, hence, it is not entirely portable.

In order to mitigate the checkpointing overhead, incremental checkpointing technique is used [96] 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 [96, 97].

Checkpointing of distributed applications is more complicated. For these applications, not only the state of the application on each running node has to 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 [98]. 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 [99], BLCR [94], MPICHV [100] 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 [100], the connection amongst 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.

Checkpointing in VM-based Systems

Virtualisation technique provides application-transparent checkpointing as an inherent feature that involves saving (suspending) and restoring (resuming) of the VM state [101–103].

In a virtualised platform, the hypervisor 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. The disk image snapshot also has to be stored, especially when the checkpointed VM is transferred and sharing the image is not possible. Current hypervisors, 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 disk size [15]. 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 [104]. 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 execution of 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 2.7). In the next step, checkpointing protocols save the messages in transit (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 [12].

2.6 An Investigation of Existing Works

In this section, we study various distributed systems from the resource contention perspective and the way their resource management systems handle the contention. Particularly, we review contention management in Clusters, Grids, and Clouds. We identify and categorise properties of the reviewed systems and summarise them in Table 2.2 for Clusters and in Table 2.3 for Grids and Clouds.

2.6.1 Contention Management in Compute Clusters

Compute Clusters are broadly categorised 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 [105]. From the resource contention perspective, shared Clusters are generally prone to request-initiated contention.

A Virtual Cluster is another variation of Clusters that work based on VMs and recently has attracted many users and providers. Although users of these Clusters are given root access to the VMs, these resources are not dedicated to one user in hardware level because several VMs on the same node can be allocated to different users.

A Multi-Cluster is an interconnected distributed system that consists of several Clusters possibly belonging to different organisations. Multi-Clusters are prone to origin-initiated contentions as well as request-initiated contention.

Shirako [106], 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 with tickets that are redeemable at the provider Cluster. In fact, Shirako broker handles inter-domain-initiated contentions by coordinating resource allocation across different Clusters. However, the user application should decide how and when to use the resources.

VioCluster [9], 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. The machine broker also implement policies for reclaiming resources that react to origin-initiated contention by preemption of a leased VM to another domain. A 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 [13], is a lease manager that is able to schedule a combination of advance-reservation, best-effort, and immediate leases. Haizea acts as a scheduling back-end for OpenNebula [36]. The advantage of Haizea is considering and scheduling the preparation overhead of deploying VM disk images. For scheduling advance-reservation and Immediate leases, leases with lower priority (i.e., best-effort) are suspended and resumed after the reservation is finished. In fact, Haizea provides a reactive resource contention mechanism for request-initiated contentions where requests have diverse QoS constraints.

Sharc [105] 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 [11] (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 approach 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 [107] provides services to the clients based on the notion

of service class (partitioning). This is performed by allocation of resource partitions to parallel applications and dynamically adjusting the partitions on each node based on the user demand. Indeed, Cluster Reserve applies the partitioning scheme to cope with request-initiated contention problems. The resource management problem is considered as a constrained optimisation problem where the inputs of the problem are periodically updated based on the resource consumption.

Muse [108], 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 optimisation problem that maximises the overall profit. Muse considers energy as a driving issue in resource management of server Clusters.

MUSCLE [109] is an off-line, global scheduler for multi-Clusters 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 minimised make span and idle times.

Lee et al. [110] proposed a global and a local scheduler for a multi-Cluster. The local scheduler is a variant of the backfilling that grants priority to jobs that need many nodes 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. [42] applied an admission control policy for shared Clusters. Such approach causes 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 utilisation of resources. Consumers also bid for 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.

Table 2.2: Classification of contention management approaches in the existing Cluster systems.

System	**Provisioning Model**	**Operational Model**	**Context**	**Contention Initiation**	**Contention Management**	**RMS Component**
Haizea [13]	Lease	Reactive	Cluster	Request	Preemption	Local Sched
VioCluster [9]	VM	Proactive & Reactive	MultiCluster	Origin	Preemption	Local Sched
Snell et al. [10]	Job	Reactive	Cluster	Request	Preemption	Local Sched
Lawson et al. [4]	Job	Proactive & Reactive	Cluster	Request	Partitioning	Local Sched
Walters et al. [38]	VM	Reactive	Cluster	Request	Preemption	Local Sched
Scojo-PECT [81]	VM	Reactive	Cluster	Request	DiffServ & Preemption	Local Sched
MOSIX [80]	VM	Reactive	Cluster	Origin	Fairness & (Preemption)	Local Sched
Sharc [105]	Job	Reactive	Cluster	Request	Partitioning	Local Sched & Admission Ctrl
COD [11]	Lease	Reactive	Cluster	Request	Partial Preemption	Local Sched
Cluster Reserves [107]	Job	Proactive	Cluster	Request	Partitioning	Local Sched
Muse [108]	Job	Reactive	Cluster	Request	Economic (Utility)	Local Sched
Shirako [106]	Lease	Reactive	MultiCluster	Inter-domain	Token	Local Sched
Lee et al. [110]	Job	Proactive & Reactive	MultiCluster	Request	Fairness	Global & Local Sched
MUSCLE [109]	Job	Proactive	MultiCluster	Request	Utility	Global Sched
Percival et al. [42]	Job	Reactive	Cluster	Request	Economic (Utility)	Admission Ctrl

2.6.2 Contention Management in Desktop Grids

This form of distributed computing, which is also known as volunteer computing, inherently relies on participation of resources, mainly Personal Computers. In desktop Grids resources 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 [111] is a famous desktop Grid project that works based on the BOINC [112] software platform 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, by utilisation of 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. An instance of this approach is HP's I-cluster [113, 114]. However, this approach can potentially interrupt interactive users (owners). Therefore, idle cycle prediction has to be done conservatively to avoid interrupting the interactive user. Both of these approaches are heavily dependent on accurate prediction of and efficient harvesting of idle cycles. Indeed, these approaches operate efficiently where there are huge idle cycles.

Recently, the VM technology has been employed in desktop Grids. The advantages of using VMs in these environments are three-fold. 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 lightweight processes, can be harvested.

NDDE [6] is a platform that utilises VMs to exploit idle cycles for Grid or Cluster usage in corporations and educational institutions. This system is able to utilise idle cycles that appear even while the user is interacting with the computer. Indeed, in this system guest and owner applications run concurrently. This approach increases the harvested idle cycle to as many as possible with minor

impact on the interactive user's applications. The NDDE has more priority than idle process in the host operating system. Therefore, it will 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 belonging to NDDE are preempted and changed to "ready-to-run" state.

Fine-grained cycle sharing system (FGCS) [8] 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 migrated off the resource because of a local request;
2. Host suddenly discontinue contributing resource to the system.

To cope with these problems, they define unavailability times in the form of a state diagram where each state is a condition that resource becomes unavailable (e.g., contention between users, and host unavailability). The authors 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.

2.6.3 Contention Management in Computational Grids/Grid Federations

Grids were initially structured based on the idea of the virtual organisations (VOs). A VO is a set of users from different organisations who collaborate towards a common objective. Several organisations constitute a VO by contributing a 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 organisation gets access to the VO resources according to the amount of resources it offers to the VO.

Organisations usually retain part of their resources for their organisational (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 contention [115]. Particularly, inter-domain-initiated resource contention arises when organisations 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 organisation. Finally, request-initiated contention exists between different types of requests (short/long, parallel/serial, and etc.).

Gruber/Di-Gruber [116] is a Grid broker that deals with the problem of resource procurement from several VOs and assigns them to different user groups. Gruber provides monitoring facilities that can be used for inter-domain-initiated contention. It also manages the enforcement 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 outsourcing of jobs to a suitable site (site selector component). Di-Gruber is the distributed version of Gruber that supports multiple decision points.

InterGrid [1] 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., organisational/local users) as well as users coming from other Grids (external). InterGrid is prone to origin-initiated (between local and external requests) and inter-domain-initiated (between different Grids) resource contention.

Peering arrangements between Grids coordinate exchange of resources and functions based on peer-to-peer relations established amongst Grids. Each peer is built upon a pre-defined contract between Grids and handles inter-domain-initiated contentions between the two Grids. Outsourcing unit of InterGrid is incorporated in the scheduling and determines when to outsource a request to a public Cloud provider.

Delegated-matchmaking [78], proposes an architecture which 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 is delegated in different sites of Grids. From the resource contention perspective, the matchmaking mechanism is in charge of dealing with request-initiated contentions through an outsourcing scheme.

Li [5], analyzed the load distribution problem in a Cluster, that is a resource provider of a Grid, and origin-initiated contention takes place between local (dedicated) and external (generic) requests. He applied a preemption mechanism to resolve the contention and proposed a probabilistic scheduling policy for the Cluster that determines the probability of sending external jobs to each Cluster node. The aim of the scheduling policy is to minimise the response time of external requests.

GridWay [20], 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 receive the required processing power or the job waiting time is more than an appointed threshold. In this situation, GridWay migrates (outsource) the job to another Grid in order to provide the demanded resources to the job. We can consider GridWay as a global scheduler that deals with request-initiated resource contentions.

OurGrid [117] is a Grid that operates based on a P2P network between sites and share resources based on reciprocity. OurGrid uses network of favours as the resource exchange scheme between participants. According to this network, each favour to a consumer should be reciprocated by the consumer site at a later time. The more favour participants do, the more reward they expect. From the resource contention perspective, OurGrid uses incentive-based approach to figure out the problem of inter-domain-initiated contentions in a Grid.

Sandholm et al. [52] investigated how admission control can increase user fulfilment in a computational market. Specifically, they considered the mixture of best-effort (to improve resource utilisation) and QoS-constrained requests (to improve revenue) within a virtualised 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.

2.6.4 Contention Management in Computational Clouds

Advances in virtual machine and network technologies led to the establishment of commercial providers that offer numerous computational 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 [118]. There are various models for delivery 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 broadly categorised as public, private, and hybrid Clouds [119]. To cope with the shortage of resource availability, particularly in private Clouds, the idea of federated Cloud has been presented [118]. Cloud federation is a possible solution for a Cloud provider in order to obtain access to a larger pool of resources.

Similar to Grid environments, Clouds are also prone to different types of resource contention. However, as Clouds are more profit-oriented in comparison

to Grids, the resource contentions solutions are also mostly commercially driven.

Amazon offers spot instances to sell the unused capacity of their data centres [120]. 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 terminated or alternatively suspended 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 et al. [120] evaluated dynamic checkpointing schemes in such a market, which is adaptive to the current instance price, and achieves cost efficiency and reliability when utilising spot instances.

Van et al. [121] proposed a multi-layer, contention-aware resource management system for Cloud infrastructures. 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 include 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 VMs.

RESERVOIR [122] is a research initiative that aims at developing the technologies required to address 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 [118] 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 [123]. Toosi et al. [124] 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 allocates resource to that via termination of 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.

Table 2.3: Classification of contention management approaches in the existing Grid and Cloud systems.

System	**Provisioning Model**	**Operational Model**	**Context**	**Contention Initiation**	**Contention Management**	**RMS Component**
GridWay [20]	Job	Reactive	Grid Federation	Request	Outsourcing	Global sched (Outsourcing)
Amazon Spot Market	VM	Reactive	Cloud	Request	Economic (Auction)	Global sched
Van et al. [121]	VM	Reactive	Cloud	Request	Economic (Utility)	Global sched
OurGrid [117]	Job	Reactive	Grid	Inter-domain	Incentive	Global sched
Ren et al. [8]	Job	Proactive	Desktop Grid	Origin	Preemption	Local sched
Li [5]	Job	Proactive	Grid	Origin	Preemption	Local sched
InterGrid [1]	Lease	Proactive	Grid Federation	Origin	Partitioning	Global sched
InterCloud [124]	VM	Reactive	Cloud Federation	Request	Economic (Utility)	Admission ctrl
RESERVOIR [122]	VM	Reactive	Cloud Federation	Request	Outsourcing	Outsourcing
Sandholm et al. [52]	VM	Reactive	Grid	Request	Partial Preemption	Admission ctrl
InterGrid Peering [1]	Lease	Reactive	Grid Federation	Inter-domain	Global scheduling	Global sched
NDDE [6]	VM	Reactive	Desktop Grid	Origin	Preemption	Local sched
Gong et al. [7]	Job	Proactive	NOW	Origin	Preemption	Local sched
Delegated-matchmaking [78]	Job	Reactive	Grid Federation	Request	Outsourcing	Outsourcing
Gruber [116]	Job	Reactive	Grid	Inter-domain & Request	Global scheduling & Outsourcing	Global sched & Outsourcing

2.7 Positioning of this Thesis

A solution for resource contention in an interconnected distributed system requires strategies to avoid contentious situation in a way that the number of resource contentions decreases to the minimum possible. As there are always circumstances that resource contention takes place, the solution requires remedial strategies to handle the contentious situation. As discussed in this chapter, these strategies can be implemented in different components of resource management systems.

This dissertation proposes a solution for origin-initiated contention in InterGrid, as a platform that enables resource sharing between computational Grids, and meets the aforementioned requirements. InterGrid employs VM-based leases as the resource provisioning model.

The idea of InterGrid is inspired by the manner Internet Service Providers (ISPs) establish peering arrangements in the Internet [2]. Similar arrangements are established between InterGrid Gateways (IGGs) in InterGrid that facilitate resource sharing between Grids. IGGs are considered global schedulers (meta-scheduler) for each Grid. They handle resource sharing with other peer Grids based on peering agreements. Additionally, provisioning rights over resource providers (RPs) within each Grid are delegated to the IGG which enables it to schedule arriving external requests on the RPs. External requests scheduled in an RP contend with local requests of the RP to gain access to the computational resources. Therefore, RPs in InterGrid are prone to origin-initiated resource contention between local and external requests.

Similar to VioCluster [9] and Haizea [13], we consider VM preemption mechanism to resolve the origin-initiated resource contention. This mechanism stops the running request and frees the resources for the higher priority request. Later, the preempted request can resume its execution from the preempted point. Specifically, preemption is an appropriate mechanism, due to using VM technology in InterGrid and the VMs'ability to suspend, resume, and migrate without affecting the computation inside them.

We introduce the preemption of external requests in favour of local requests in InterGrid when there are not sufficient resources to serve local requests. We consider the side-effects of preempting VM-based leases in terms of the imposed overhead time, number of resource contention, and response time for external requests. Different proposed strategies in this dissertation operate within various components of the InterGrid platform and try to handle these side-effects.

With regard to the imposed overhead time for preempting VMs, we investi-

gated and modelled the overhead time based on various possible operations that can be performed on VMs. Our proposed model extends the existing model of the Haizea [17] scheduler and considers communication between the VMs of a lease. Another extension in comparison to the model proposed by Sotomayor et al. [17] is that we model the overhead of migrating VMs in addition to suspending and resuming.

We propose a preemption policy in the local scheduler of the RPs that is aware of resource contention between requests. The policy proactively selects a set of leases for preemption in a way that the resource contention decreases. An idea similar to the preemption policy was investigated by Snell et al. [10], however, they have not taken into account the overhead side-effect in their decision making. Indeed, they terminate the job for preemption, which reduces the imposed overhead to zero.

We propose a contention-aware scheduling in IGG, that aims at avoiding resource contention via scheduling of arriving external requests on different resource providers within its domain. It is an on-line scheduling policy that proactively schedules the external requests with regard to the local workload characteristics, such as arrival rate, in each RP. The scheduling policy also considers the situation that some external requests are more valuable and reduces the likelihood of contention for them. The most similar research work to this scheduling policy is the one proposed by Li [5]. Nonetheless, Li's scheduling goal is to minimise the response time of external requests whereas our goal is to minimise the overall number of resource contentions. The other significant difference is that Li has solved the problem for the local scheduler of a Cluster whereas our scheduling policy works in the global scheduler of a Grid.

As the occurrence of origin-initiated contention is inevitable, we propose remedial strategies that react to the contentions and try to handle their side-effects, such as long response time for low priority requests. More specifically, we propose a contention-aware policy in the admission control unit of the resource providers in InterGrid. The policy reduces the impact of preemption mechanism on the response time of external requests and prevents long response time for them. The admission control policy works based on limiting the queue length for external requests in a way that their response time would be limited. According to the policy, external requests are accepted until their predicted response time is less than a threshold value.

The proposed policy is different from the one proposed by Sandholm et al. [52] in the sense that it performs the feasibility test for each arriving request. Conversely, our proposed policy finds out the ideal queue length and does not impose

overhead to the system for each arriving request. Although our scenario share similarities with the research undertaken by Gong et al. [7], the main difference is that we consider several parallel external requests whereas their policy handles one sequential external request that is allocated to a node. In other words, they do not consider queuing external requests, whereas our policy concentrates on finding the number of external requests that can be accepted while their response time is limited.

Recently, many research works were conducted on energy-aware management of resources in a provider. Energy management policies commonly utilise resource consolidation mechanism and switch off lightly loaded resources in order to preserve energy. However, reducing the number of active resources in a contention-prone system raises the likelihood of contentions within RPs. More importantly, an aggressive energy saving policy that switches on few resources can lead to long response time for external (low priority) requests.

We propose a contention-aware energy management policy for resource providers in InterGrid that employs consolidation to save energy while it considers the contention side-effect for external requests. Specifically, the policy tries to minimise the energy consumption of resource providers while external requests can be served within a limited response time. Based on the classifications provided in this chapter, the proposed contention-aware energy management policy reacts to the contentious situation to handle its side-effect in terms of response time. Additionally, this policy is applied in the resource provider level along with the local scheduler.

2.8 Summary

Focusing on interconnected distributed systems, in this chapter we introduced different possible types of resource contentions along with common approaches to resolve them. Then, we investigated the potential role of different components in a resource management system to resolve various types of contention. Particularly, we recognised the role of resource provisioning model, local scheduling, global scheduling, and admission control unit in a resource management system.

We realised that the emergence of VM-based resource provisioning model has posed the preemption mechanism as a predominant solution for different types of resource contention. Therefore, in this chapter, we also dealt with the challenges and opportunities of preempting VMs. We reviewed existing approaches in Clusters, Grids, and Clouds from the contention management perspective and categorised 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 where the contention is considered. We also presented the requirements for handling resource contention between distributed systems and positioned this dissertation with regards to existing works.

In the next chapters, we describe the contention management strategies, aiming to provide resources to external requests without impacting the performance of providers local users.

Chapter 3

Preemption-based Contention Management in InterGrid

The contention problem in InterGrid is resolved with preemption of requests from external users in favour of local users'requests. However, preemption of VM-based requests entails side-effects in terms of time overhead and long response time for external requests. In this chapter, we model the imposed overhead of preempting VMs based on different operations on them. Then, we propose and compare preemption policies that determine the proper set of request(s) for preemption in a way that the side-effects are reduced.

3.1 Introduction

As discussed in Chapter 2, and shown in Figure 3.1, in InterGrid computational resources in each RP are shared between external and local users. Hence, resource provisioning in InterGrid is performed for two types of users, namely: local users and external users. As illustrated in Figure 3.1, local users refer to users who ask their local RP for resources by submitting a *local request* to the Local Resource Management System (LRMS). External users send *external requests* to IGG to access larger amount of resources through resources from other Grids.

Therefore, there is a mixture of local and external requests in an RP that try to access resources. This scenario leads to origin-initiated resource contention within the RP. In this situation, the RP considers a higher priority for its local requests than external requests [125]. In other words, the organisation that owns the resources would like to ensure that its community has priority access to the resources. In this circumstance, external requests are welcome to use resources if they are available. Nonetheless, external requests should not delay the execution

of local requests.

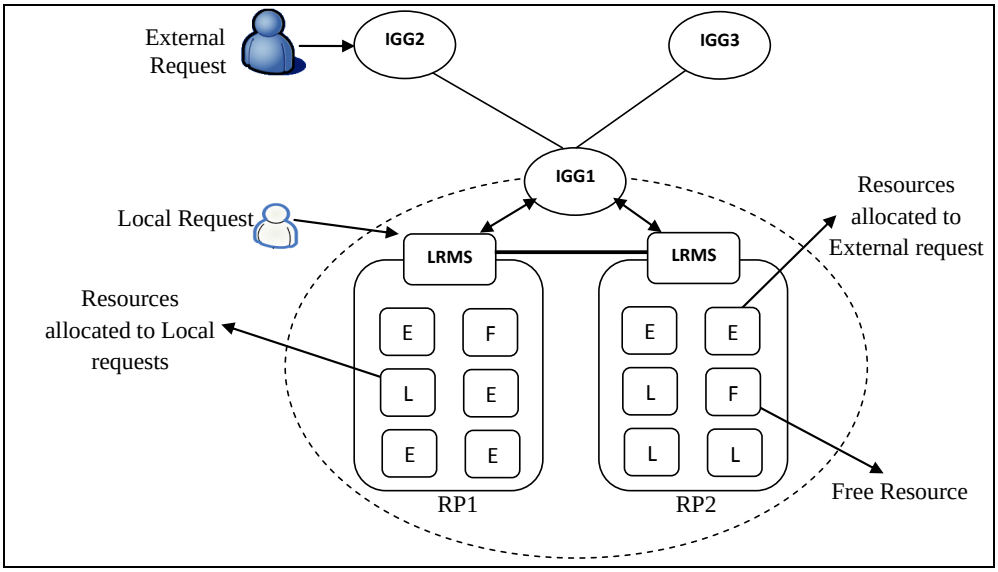


Figure 3.1: A scenario that shows 2 resource providers (RP1 and RP2) within a Grid of InterGrid. The scenario also depicts the position of local and external requests.

In this chapter, we deal with the problem of origin-initiated resource contention between local and external requests in each RP of InterGrid. More specifically, the problem we are dealing with is resource procurement for local requests when existing resources have been allocated to external requests and the available resources are not adequate to serve the local requests. In this situation, one solution is to preempt external leases and allocate the resources to local requests. However, preempting VM-based leases involves 2 main side-effects.

The first side-effect is the time overhead imposed on the system for preemption of VM-based leases. The overhead varies based on the type of operation performed on the VMs. For instance, the overhead of VM suspension is lower than the overhead of VM migration. The imposed overhead of VM preemption can affect the resource utilisation of an RP. This impact is particularly remarkable when the arrival rate of local requests is high [78]. Additionally, a precise estimation of the imposed overhead is necessary for implementation of preemption-based resource scheduling [17].

Existing works on modelling the time overhead of VM preemption [17] consider the amount of memory that should be de-allocated, the number of VMs mapped to each physical node, and the use of local or shared storage. However, there are other factors such as communication between VMs of a lease that also have to be considered. Therefore, one problem we address in this chapter is to model the overhead time of preempting VM-based leases by taking into account

communication between VMs.

The second side-effect of lease preemption is on increase in the waiting time and consequently response time of external requests. Indeed, many of the current distributed systems use a variation of the backfilling strategy [10] for scheduling. In the this strategy, future resource availabilities are reserved for other requests that are waiting in the queue. Preemption of leases and vacation of resources for local requests can potentially affect these reservations. For instance, in Figure 3.2 vacation of resources between t_1 and t_2 affect the currently running leases (leases 1 and 4) as well as reservation waiting in the queue (leases 2, 3, and 5). The affected leases and reservations are scheduled at a later time in the scheduling queue. Therefore, preemption of external leases delays their execution and increases their response time.

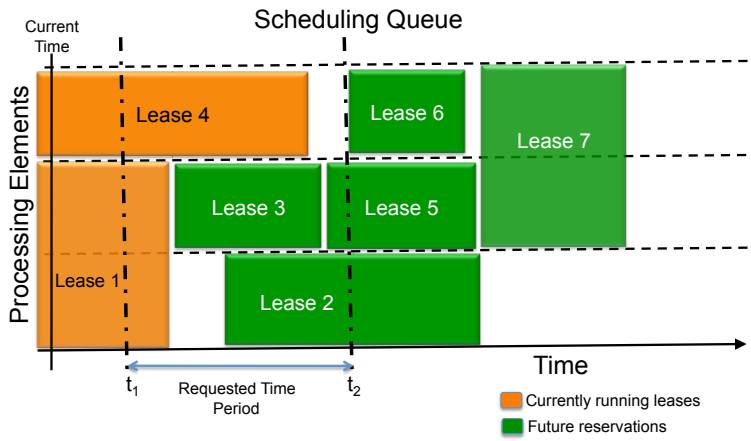


Figure 3.2: Resource contention occurs when the interval t_1 to t_2 is requested by a local request. In this situation, both running and waiting leases are affected.

When lease preemption is enabled in an RP, there is a possibility that several leases have to be preempted to create sufficient vacant resources for a local request. Therefore, there are potentially several sets of candidate leases that can be preempted. We term each set of candidate leases a *Candidate Set*. For example, in Figure 3.2 if a local request needs 1 VM for the duration of t_1 to t_2 the candidate sets are leases $\{1, 2\}, \{1, 3, 5\}, \{4\}$. Selection of different candidate sets for preemption affects the amount of imposed overhead as well as the response of external requests. Therefore, another problem we address in this chapter is how to choose an optimal candidate set for preemption in a way that the side-effects of lease preemption are reduced.

The formal definition of the problem is:

- L_i: Lease i.
- R_j: Local request j

- $\tau(L_i)$: Type of external request. As will be discussed in this chapter: $\tau(L_i) \in \{extCancellable, extSuspendable, extMigratable, extNonPreemptable, localNonPreemptable\}$
- $v(L_i)$: Number of VMs in the lease/request i.
- $h(L_i)$: Overhead for preemption of lease i.
- $p(L_i)$: Category of lease i (local or external) and defined as follows:

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

According to the above definitions, a candidate set C_m is a set of external leases that their preemption vacates enough resources for allocation of a local request R_j. If there are S candidate sets, then all candidate sets can be presented as:

$$A : \{C_m \,|\, 0 \leq m \leq S - 1\} \tag{3.1}$$

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

3.2 Proposed Solution

3.2.1 Introducing Different Lease Types

One difference between job-based resource provisioning and lease-based resource provisioning is that jobs can be preempted without notification to the user (job owner), whereas preemption of leases is restricted by definition [72].

Therefore, first we introduce regulations in the lease terms to make the lease preemption possible. For that purpose, we introduce different request (lease) types in InterGrid. A request issued by a user in InterGrid has the following information:

- Virtual Machine (VM) template requested by the user.
- Number of VMs needed.
- Ready time: the earliest time that request can be started.
- Duration of the lease.
- Deadline: the latest time for request completion.

We extend InterGrid request by adding the "request type" to it. In practice, the type of a request expresses the Quality of Service (QoS) level required by that request. The types we consider for requests in InterGrid are broadly classified as best-effort and deadline-constraint requests. More details of the different request types are as follows:

- *Best-Effort-Cancellable*: these requests can be scheduled at any time after their ready time. However, they can be canceled without notification of the lease owner. Cancellable leases neither guarantee the deadline nor the duration of the lease. Such leases are applicable for map-reduce-like requests [82]. Spot instances in Amazon EC2[1] are another example of cancellable leases.

- *Best-Effort-Suspendable*: This type guarantees the duration of the lease but not within a specific deadline. Suspendable leases are flexible in start time and they can be scheduled at any time after their ready time. In the case of preemption, these leases are suspended and then rescheduled in another free time-slot to resume their execution. Suspendable leases are suitable for Bag-of-task (BOT) and Parameter Sweep applications [126].

- *Deadline-Constraint-Migratable*: This type guarantees both the duration and the deadline for a lease. However, there is no guarantee that the lease will be run on a specific resource(s). In other words, there is always a chance for the lease to be preempted but it will be resumed and finished before its deadline, either on the same resource or on another resource. Migratable leases are needed by steerable applications [127]. In these applications, the workload can be migrated to more powerful RPs to meet user constraints such as deadline [127].

- *Deadline-Constraint-Non-Preemptable*: The leases associated with such requests cannot be preempted. These leases guarantee both deadline and duration without preemption during the lease. This type of lease is useful for critical tasks in workflows where some tasks have to be started and completed at exact times to prevent delaying the execution of the workflow [89]. Advance-reservation requests is another example that can be served by this type of lease.

Local requests have priority over external requests and they should not be preempted. Therefore, we consider them as non-preemptable. However, external users can make requests of any of the types mentioned above.

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

In practice, different request types imply different prices. Thus, users are motivated to associate their requests to different request types. Unarguably, the more flexible request type, the less expensive the lease. Additionally, the type of request affects the operations that can be performed on it at preemption time.

3.2.2 Measuring the Overhead of Lease Preemption

The time overhead imposed for preemption of a candidate set depends on the type of leases contained in the candidate set. In other words, the overhead is driven by the operations performed on the VMs of leases involved in a candidate set. In this part, we discuss the worst-case time overhead of possible operations on the VMs of a lease.

Preemption of *cancellable* leases imposes the minimum time overhead. This overhead concerns the time needed to stop VMs of a lease. The duration of the stop operation is independent from the VM characteristics, such as memory size of the VM, and it is almost negligible [12, 17].

Preemption of a *suspendable* lease is more complicated than a cancellable lease. One complexity relates to the message exchange between the VMs of the lease. In fact, execution of a distributed application in the VMs of a lease implies exchanging messages between the VMs. However, suspension of a lease occurs in the VM level, which is unaware of the communication. Therefore, suspension of a lease can interrupt the message communication and lead to inconsistent state of jobs running inside VMs [12].

More specifically, message passing is commonly performed through the TCP protocol to assure the message delivery. If the sender host does not receive an acknowledgement after a certain number of retransmissions, the connection with the receiver is terminated. At the suspension time, there is a possibility that some VMs become unreachable and consequently some connections are lost. Therefore, it is important to coordinate the suspension and resumption operations to avoid inconsistency for the jobs executing within VMs.

One possible way to reduce the impact of the suspend and resume operations on the jobs running within the VMs, is *pausing* them before suspension and *unpausing* VMs after resumption. Pausing a VM prevents it from accessing the processor and is supported in current hypervisors such as Xen [128]. This operation is quick (takes few milliseconds) and can be completed before the unreachable delay of TCP [12].

Therefore, the time overhead of preempting a *suspendable* lease broadly includes the time to pause its VMs, suspending (i.e., writing the memory image of

the VMs to the disk), and rescheduling the lease. Accordingly, resuming the lease includes the time for VM resumption (i.e., the time for loading the VMs'memory image from the disk), and then unpausing VMs.

Since pausing and unpausing operations take a short time, usually they are performed sequentially on VMs (e.g., based on the VM identifier) [12]. If a lease L_i contains $v(L_i)$ VMs, then the time for pausing the VMs is $v(L_i)\cdot t_p$ where t_p is the time to pause a VM. In our analysis, we consider the same amount of time for pausing and unpausing operations. Therefore, the overall time overhead of pausing and unpausing is $2v(L_i)\cdot t_p$.

Suspension of a lease requires its reschedule for the remainder of the execution. The time overhead of rescheduling depends on the time complexity of the rescheduling algorithm. The complexity usually depends on factors such as number of physical nodes and current workload condition. Since our model does not assume any particular scheduling policy, we consider a constant value (δ) for the rescheduling time overhead.

The major time overhead in suspendable leases is caused by the time for suspension and resumption of VMs. The time for these operations is driven by the memory size of each VM. Specifically, the suspension time for one VM (t_s) and resumption time (t_r) are defined based on Equations 3.2 and 3.3, respectively.

$$t_s = \frac{mem}{s} \tag{3.2}$$

$$t_r = \frac{mem}{r} \tag{3.3}$$

where mem is the memory size of VM, s is the rate of suspending megabytes of VM memory per second, and r is the rate of re-allocating megabytes of VM memory per second [17].

We consider a shared storage for an RP to be able to resume the suspended lease on any of its hosts. In this circumstance, to avoid contention on the shared storage, the suspension and resumption operations are performed sequentially for all VMs of lease. Therefore, for lease L with $v(L)$ VMs, the suspension time overhead is $\sum_{i=1}^{v(L)} t_s^i$ [17].

It is worth noting that this is the worst-case analysis for the suspension and resumption time overheads. We expect that considering other factors, such as overlapping, can result in lower overheads for these operations.

By taking into account all the above factors the overall time overhead for suspending (L^s) and resuming a lease (L^r) are calculated according to Equations 3.4

and 3.5, respectively.

$$L^s = v(L) \cdot t_p + \sum_{i=1}^{v(L)} t_s^i + \delta \tag{3.4}$$

$$L^r = v(L) \cdot t_p + \sum_{i=1}^{v(L)} t_r^i \tag{3.5}$$

where t_p is the time for pausing a VM; t_s^i is the time overhead of suspending the i-th VM of a lease; and t_r^i is the time overhead of resuming the i-th VM of a lease. Therefore, the overall time overhead for preemption of a suspendable lease ($h(L)$) is:

$$h(L) = 2v(L) \cdot t_p + \delta + \sum_{i=1}^{v(L)} (t_s^i + t_r^i) \tag{3.6}$$

The time overhead imposed for preemption of *migratable* leases includes VM image transferring overhead in addition to all overheads considered for suspendable leases [14]. More specifically, migration of a VM includes a VM suspension on the source host, transfer of the suspended VM to the destination RP, and resume of the VM in the destination host. The overheads caused by pausing, unpausing, and rescheduling of VMs (i.e., finding a proper destination) also have to be taken into account.

In the transferring phase, the disk memory image of the suspended VM is transferred to the destination host over the network [14]. The time for transferring depends on the size of the suspended VM and the network bandwidth, therefore, $t_{copy}^j = mem_j/b$ where mem_j is the size of disk memory image for VM_j and b is the network bandwidth.

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

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

The overheads regarding rescheduling, pausing, and unpausing of VMs are the same as those discussed for suspendable leases. The overall overhead for migration of VMs of lease i (L_i) is defined based on Equation 3.7.

$$h(L_i) = \sum_{j=1}^{v(L_i)} t^j_{copy} + t^1_s + \sum_{j=1}^{v(L_i)-1} max\{t^j_s, t^{j-1}_r\} + t^{v(L_i)}_r + 2v(L_i)\cdot t_p + \delta \qquad (3.7)$$

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

3.2.3 Preemption Policy

When an LRMS cannot find sufficient vacant resource to allocate an arriving local request, it forms all candidate sets where each candidate set contains leases whose preemption creates enough space for the local request. The preemption policy determines the proper candidate set for preemption.

From the user perspective, the selection of different candidate sets is decisive for the amount of resource contention within an RP between local and external requests. Additionally, preemption of various candidate sets affects the number of external leases to be preempted and their waiting times.

From a system centric perspective, the choice of different candidate sets determines the number of VMs to be preempted as well as the operation that has to be enacted on them (e.g., suspension or migration). Therefore, the choice of different candidate sets influences the amount of time overhead imposed to the system.

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

Minimum Overhead Policy (MOV)

To consider system centric metrics such as resource utilisation, this policy aims at minimising the imposed time overhead to the underlying system.

For that purpose, the total overhead imposed to the system by each candidate set is calculated based on the analysis provided in Section 3.2.2. Then, the set with minimum overhead is selected for preemption. The MOV policy formally is presented based on Equation 3.8.

$$MOV(A) = \min_{m=0}^{S-1}\{h(C_m)\} \tag{3.8}$$

Minimum Leases Involved Policy (MLIP)

MLIP is a contention-aware preemption policy that aims at minimising the number of contentions between local and external leases. For that purpose, MLIP selects the candidate set that contains the minimum number of leases for preemption. This policy disregards the type of leases involved in the candidate set.

We can consider MLIP as a user-centric policy. In fact, preemption of leases increases waiting times, hence, users do not desire that their leases be preempted. Therefore, MLIP sought to satisfy more users by preempting fewer leases.

Formally, MLIP can be presented according to Equation 3.9.

$$MLIP(A) = \min_{m=0}^{S-1}\{|C_m|\} \tag{3.9}$$

where $|C_m|$ gives the number of leases involved (cardinality) in each candidate set C_m.

Minimum Overhead Minimum Lease Policy (MOML)

The two proposed policies mentioned earlier aim to either improve resource utilisation (as a system centric criterion) or minimise the resource contention (as a user centric criterion). However, MOML policy fulfils both system and user centric criteria at the same time.

The way this policy operates is depicted in Figure 3.3 and its pseudo code is illustrated in Algorithm 1. In fact, MOML is a trade-off between MOV, which minimises the imposed overhead, and MLIP, that minimises the resource contention.

According to Figure 3.3 and Algorithm 1, in MOML the selection of a candidate set is carried out in two phases. In the first phase (pre-selection phase) all candidate sets whose total overhead smaller than a certain threshold (α) are pre-selected for the second phase (lines 5 to 8 in Algorithm 1). In fact, the pre-selection phase increases the tolerance of acceptable overhead in comparison with

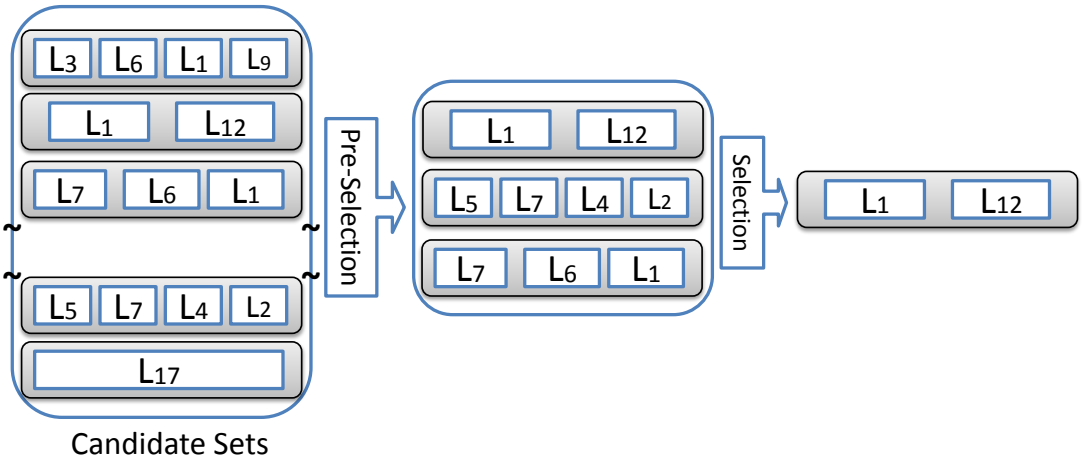


Figure 3.3: Pre-selection and final selection phases of MOML policy.

MOV. In the second phase, to have fewer resource contentions between requests, a candidate set that contains the minimum number of leases is selected (lines 9 to 11 in Algorithm 1).

Selection of a proper value for α determines the behaviour of MOML policy. Particularly, if the $\alpha \to \infty$, then MOML behaves as MLIP. On the other hand, if $\alpha \to 0$, then MOML approaches to MOV. To keep the trade-off between MOV and MLIP, we consider α as the *median* value of the overheads (lines 1, 2, and 4 in Algorithm 1). By choosing $\alpha = median$ we ensure that just half of the candidate sets that have lower overheads are considered in the second phase to have a minimum number of leases (i.e., cause minimum resource contention).

Algorithm 1: MOML Preemption Policy.

```
   Input: Candidate Sets
   Output: Selected Candidate Set
1  foreach candidateSet ∈ Candidate Sets do
2      Overheads.Add(getOverhead(candidateSet));
3  min ← ∞;
4  α ← getMedian(Overheads);
5  foreach candidateSet ∈ Candidate Sets do
6      ovhd ← getOverhead(candidateSet);
7      NoLeases ← Cardinality(candidateSet);
8      if ovhd ≤ α then
9          if NoLeases < min then
10             selected ← candidateSet;
11             min ← NoLeases;
```

3.3 Performance Evaluation

3.3.1 Performance Metrics

Local and External Requests Rejection Rate

The initial objective of this part of our research is to serve more local requests by preempting resources from external leases. Therefore, it is interesting to determine the efficiency of different preemption policies in terms of serving more local requests.

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

Additionally, we want to investigate if decreasing of local request rejection rate causes rejection of more external requests. External request rejection rate describes this metric and shows the percentage of external requests that are rejected. The ideal case is that local request rejection rate is reduced without increasing the external request rejection rate.

Resource Utilisation

Time overhead is a side-effect of VM preemption that degrades resource utilisation. Therefore, we investigate how different preemption policies affect the resource utilisation. Resource utilisation is defined according to the Equation 3.10.

$$Utilisation = \frac{computationTime}{totalTime} * 100 \tag{3.10}$$

where:

$$computationTime = \sum_{i=1}^{|\lambda|} v(L_i) \cdot d(L_i) \tag{3.11}$$

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

Number of Lease Preemptions (Resource Contention)

As preemption is the consequence of resource contention, the total number of lease preemptions is a proper metric to measure the resource contention. This metric presents user satisfaction resulted from different preemption policies.

Response Time

Response time is a user-centric metric that is affected by preemption. This metric is prominent for best-effort external requests that are in the risk of getting preempted several times that increases their response time. This metric measures the amount of time on-average a best-effort lease should wait beyond its ready time to be completed. Average response time of best-effort external requests (ART) is calculated based on Equation 3.12.

$$ART = \frac{\sum_{L \in \beta} (c_L - s_L)}{|\beta|} \tag{3.12}$$

where, β is the set of best-effort external leases and $|\beta|$ is the number of leases in this set. c_l and s_l show completion time and ready time of lease L, respectively. Although best-effort requests are not bound to any deadline, users are more satisfied to wait less for their requests to be completed.

3.3.2 Experimental Setup

For simulation, we used GridSim [129] as a discrete event simulator. In the experiments, Lublin99 [130] has been configured to generate a two-week-long workload that includes 3000 parallel requests.

Lublin99 is a workload model based on the San Diego Super Computer (SDSC) Blue Horizon machine. Job traces collected from this supercomputer are publicly available and have been studied extensively in the past.

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

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

Based on the research by Sotomayor et al. [17] and considering the 100 Mbps network bandwidth, the suspending rate of VM memory is $s = 6.36$ MB/second,

and the re-allocating rate is $r = 8.12$ MB/second (see Section 3.2.2). Hence, in our experiments, suspension time (t_s) and resumption time (t_r) for a lease with 1 VM are 161.0 and 126.1 seconds, respectively. During the migration of a VM with similar configuration, the time overhead for transferring a suspended VM to another RP (t_{copy} in Equation 3.7) of InterGrid is 160.2 seconds [14]. Based on our experiments, pausing and unpausing operations on each VM takes 5 milliseconds. Finally, we employ a conservative backfilling as the scheduling policy in the LRMS. During the initial experiments, we noticed that the average overhead time of rescheduling is 2.3 seconds.

We study the behaviour of different policies when they face workloads with different characteristics. For this purpose, we modified the characteristics of the workloads when:

- The number of best-effort external requests (i.e., Cancellable and Suspendable) varies.
- The number of deadline-constraint external requests (i.e., Migratable and Non-Preemptable) varies.
- The number of local requests varies.

Since the Lublin workload does not provide information about request types, we generated these types uniformly and assigned them to the generated workloads. We changed the percentage of best-effort and deadline-constraint requests from 10% to 50% of the external requests while the number of local requests remains constant (1000). In another configuration, local requests are changed from 20% to 70% of the whole workload. In fact, we experimented conditions where local requests are below and above these limits. However, we noticed that not many preemptions occur in those points, therefore, there is no major difference between policies. To have a realistic evaluation, in the Lublin workload we adjusted the average number of VMs to 4 and the average duration of requests 2 hours, which are obtained based on default values of parameters in the workload.

3.3.3 Experimental Results

Local and External Request Rejection Rate

In Table 3.1, the mean difference of decrease in local requests rejection rate is reported along with a 95% confidence interval of the difference. We report the difference between rejection rate in two situations; First, when no preemption policy is in place, and second, when the MOML policy is used as the preemption

policy. We use a T-test to determine the mean difference between these two policies. To perform the T-test we have ensured that the distribution of differences is normal.

According to Table 3.1, local request rejection rate significantly decreased statistically and practically by applying preemption in all cases. More importantly, this reduction in the local request rejection rate was achieved without rejection of more external requests. Based on Table 3.1, external request rejection rate does not change significantly in any of the experiments.

Table 3.1: Mean difference and 95% confidence interval (CI) of decrease in local requests rejection rate and external requests rejection rate as a result of lease preemption in an RP of InterGrid.

Modified Parameter	**Mean Decrease in Local Requests Rejection Rate**	**CI of Decrease in Local Requests Rejection Rate**	**Change in External Requests Rejection Rate**
Percentage of BE External Requests	**72.0**%	(51.1,92.8), P-Value=0.001	Not statistically significant, P-Value=0.6
Percentage of DC External Requests	54.3%	(35.0,73.7), P-Value=0.001	Not statistically significant, P-Value=0.3
Percentage of Local Requests	58.2%	(40.3,75.9), P-Value<0.001	Not statistically significant, P-Value=0.6

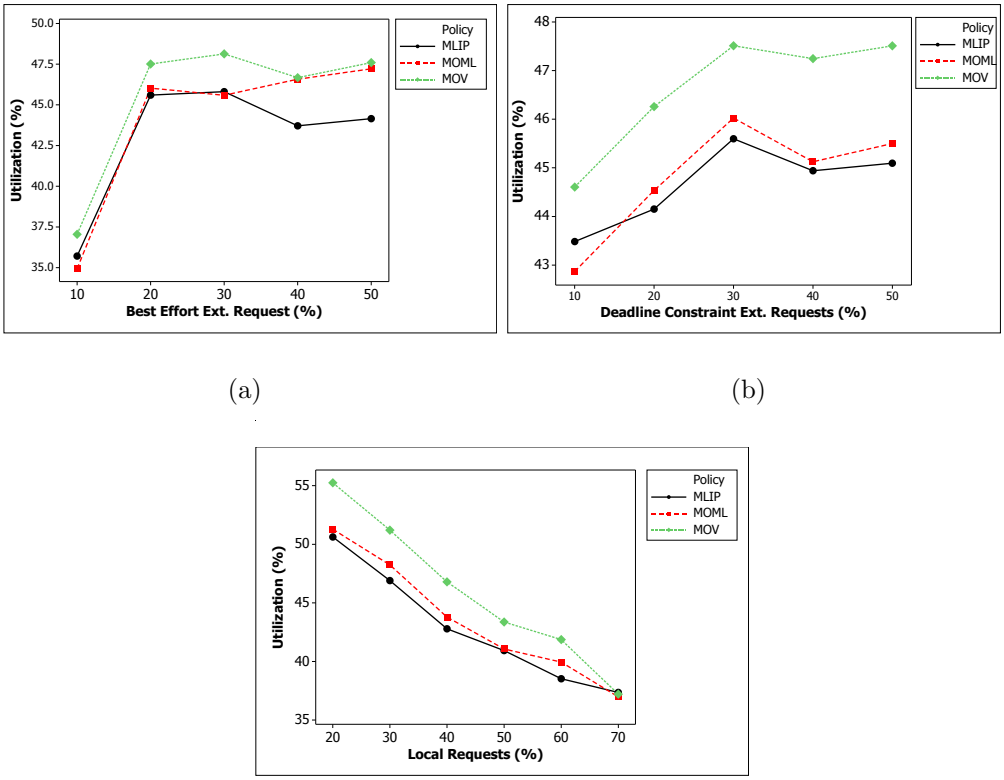
Based on this experiment, the maximum reduction in the local request rejection rate occurs when the percentage of best-effort external requests is higher (the first row in Table 3.1). In this circumstance, more local requests can be accommodated with preemption of best-effort leases.

Resource Utilisation

In this experiment, we measure the resources utilisation when different preemption policies are applied.

In all sub-figures of Figure 3.4, it is observed that the MOV policy results in better utilisation comparing with the other policies. However, in a few points (e.g., in Figure 3.4(a) when 40% of the requests are best-effort), MOV has slightly less utilisation than MOML. The reason is resource fragmentations (i.e., unused spaces) in the scheduling queue, which leads to lower resource utilisation. Subfigures of Figure 3.4 also demonstrates that the resource utilisation MOML lies between MLIP and MOV.

Figure 3.4(a) indicates that increasing the percentage of best-effort requests improves the resource utilisation; however, after a certain point (i.e., best-effort>20%) resource utilisation does not fluctuate significantly in different policies. Indeed, in this situation unused spaces are allocated to the preempted leases.



(a) (b)

(c)

Figure 3.4: Resource utilisation results from different policies. The experiment was carried out by modifying (a) the percentage of best-effort external requests, (b) the percentage of deadline-constraint external requests, and (c) percentage of local requests.

In Figure 3.4(b) shows that resource utilisation increases by increasing the percentage of deadline-constraint requests in all policies. In fact, more deadline-constraint requests imply fewer preemptions and more resource utilisation. As expected, the MOV policy outperforms other policies due to preemption of leases that impose less overhead.

In Figure 3.4(c), it is expressed that by increasing the percentage of local requests, the number of preemption and subsequently the amount of overhead is increased. Therefore, resource utilisation decreases almost linearly in all policies. Another reason for the reduction in resource utilisation is that local requests are not preemptable and their scheduling leads to many fragmentations in the scheduling queue.

Number of Lease Preemptions (Resource Contention)

The number of external leases that are preempted in different preemption policies indicates the amount of resource contention in the system.

Figure 3.5(a) shows that when the percentage of best-effort requests increases, the number of preemptions rises almost linearly. For the lower percentages of best-effort external requests (best-effort<30%), MOML behaves similarly to MOV, however, after that point MOML approaches MLIP. The reason is that when the percentage of best-effort leases is high, the likelihood of having a candidate set with the minimum number of leases and not large overall overhead is high. Thus, MOML approaches MLIP.

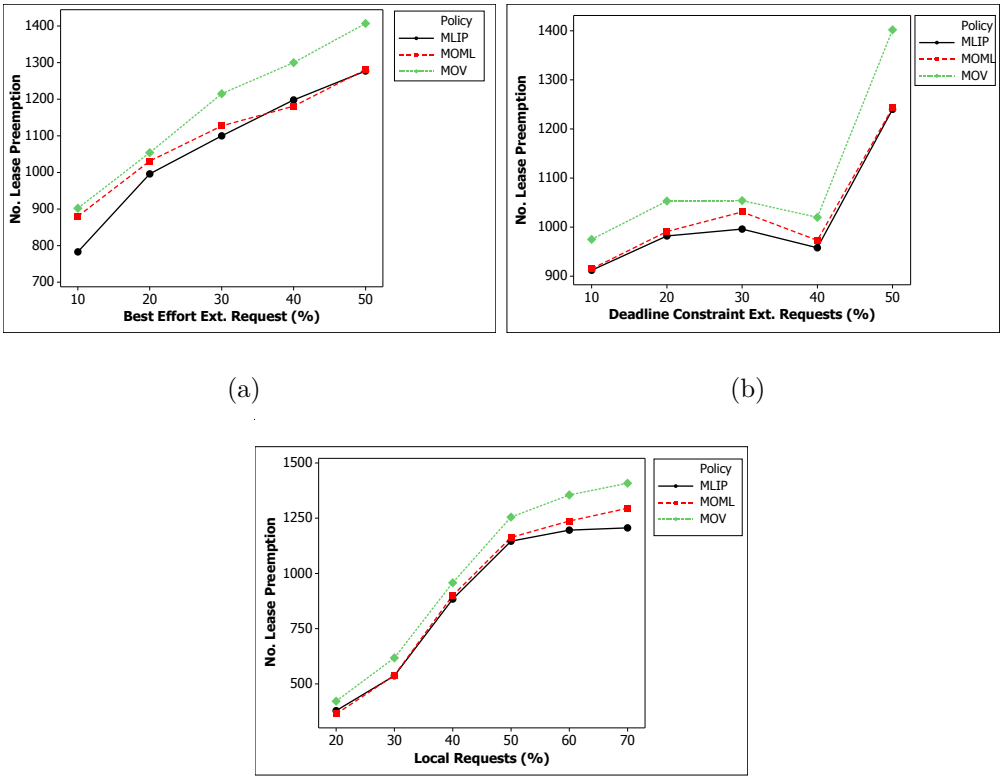


Figure 3.5: Number of lease preemption resulted from different policies by changing (a) percentage of best-effort external requests, (b) percentage of deadline-constraint external requests, and (c) percentage of local requests.

Figure 3.5(b) demonstrates that the number of preemptions does not vary significantly when the percentage of deadline-constraint requests is less than 40%. In fact, in this situation there is enough best-effort requests for preemption and changes in the percentage of deadline-constraint requests does not play an important role.

Figure 3.5(c) reveals the impact of number of local requests on the resource contention. It shows that in all policies the number of lease preemptions is increased almost linearly with the increase in the percentage of local requests.

In general, in all sub-figures of Figure 3.5, MLIP results in fewer number of

lease preemptions (resource contention) and MOML operates between MLIP and MOV.

Average Response Time

In this experiment, we investigate the impact of different preemption policies on the average response time of best-effort external requests. The results of the experiment under different workloads are illustrated in Figure 3.6.

All subfigures of Figure 3.6 show that MLIP leads to smaller response time in comparison to other policies. The reason is that MLIP disregards the type of leases for preemption. This means that, comparing with MOV, it is less likely that MLIP will preempt best-effort requests. Therefore, the best-effort requests are completed earlier and their average response time is lower in MLIP.

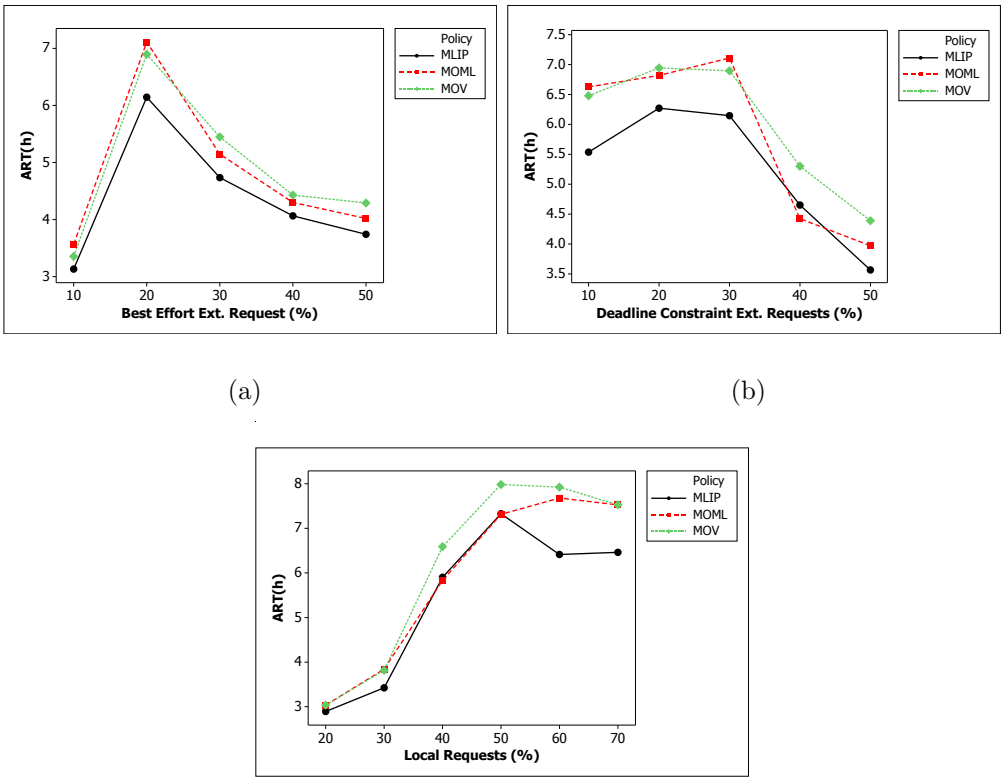


Figure 3.6: Average response time (ART) resulted from different policies. The experiment is carried out by altering (a) percentage of best-effort external requests, (b) percentage of deadline-constraint external requests, and (c) percentage of local requests.

Figure 3.6(a) demonstrates that, by increasing the percentage of best-effort requests, the average response time decreases after a certain point. When 20%

of external requests are best-effort, the average response time reaches its peak because of numerous preemptions occur. However, after that point we notice a decrease in average response time of best-effort requests. This decrease occurs due to fewer deadline-constraint requests and more opportunities for local requests to be allocated. When 10% of the external requests are best-effort, since there are not many preemptable requests in the system, many local requests are rejected and few preemption occurs. Hence, the average response time is low in that point.

Figure 3.6(b) shows that, by increasing the percentage of deadline-constraint requests, the average response time decreases. In fact, increasing the percentage of deadline-constraint requests implies fewer best-effort external requests in the system. Therefore, the average response time for best-effort external requests decreases.

Figure 3.6(c) illustrates that, by increasing the percentage of local requests in the system (and consequently increasing the number of preemptions), the average response time increases. However, the reason for stable situation in ART, when local requests are more than 50%, is that there are many local requests in the system that collide and rejected. Therefore, the number of local requests after that point does not vary significantly and the impact on ART is not substantial.

3.4 Summary

In this chapter, we investigated how origin-initiated resource contention between local and external requests can be resolved in the local scheduler of a resource provider (RP) in InterGrid. For this purpose, we applied preemption mechanism to preempt external leases in favour of local requests. We observed that preemption of leases substantially decreases the rejection of local requests (up to 72%) without increasing external requests rejection rate. Furthermore, we investigated the side-effects of the preemption mechanism when VMs are utilised for resource provisioning. Specifically, we modelled the overhead of suspension and migration operations on VMs of leases.

Then, we proposed 3 policies to decide which lease(s) are better choices for preemption. The MOV policy aims at minimising the imposed overhead time and improving resource utilisation. The MLIP policy results in less resource contention and increases user satisfaction. However, it does not lead to a high resource utilisation. Finally, the MOML policy makes a trade-off between resource utilisation and resource contention.

This chapter tackles the problem of resolving resource contention using pre-

emption mechanism at the local scheduler level through preemption policies. In the next chapter, we investigate how resource contention can be avoided by proactive scheduling of external requests in the meta-scheduler level of InterGrid (i.e., in the IGG).

Chapter 4

Proactive Resource Contention Avoidance in InterGrid Gateway

In this chapter we focus on the question of how resource contention can be avoided or reduced to the minimum possible in a Grid. We consider the problem in a scenario where some external requests are more valuable than others. Therefore, another research question answered in this chapter is how we can further decrease the likelihood of contention and preemption for the valuable external requests. To address these questions, we propose a proactive scheduling policy in the InterGrid Gateway (IGG) that reduces the amount of resource contention. Additionally, the scheduler dispatches external requests to RPs in a way that less resource contention occurs for valuable external requests.

4.1 Introduction

In Chapter 3, we demonstrated how resource contention between local and external requests can be resolved with preemption of external requests in favour of local requests. However, the side-effects of the preemption mechanism is twofold:

- From the system's owner perspective, VM preemption imposes overhead to the underlying system and degrades resource utilisation [13].
- From the external user perspective, preemption of leases causes resource contention and increases the response time of the requests.

As a result, both resource owner (who prefers to increase resource utilisation) and external users (who are interested in less contention and shorter response time) benefit from fewer contention and preemptions in the system. Therefore,

one objective of this research is to decrease the number of preemptions that take place in a Grid environment.

The objective becomes challenging further when external requests have different levels of Quality of Service (QoS) requirements (also termed different request types in this chapter). For instance, some external requests have deadlines whereas others do not. Preemption affects the QoS constraints of such requests. This implies that some external requests are *more valuable* than others, therefore, more precedence should be given to valuable requests by reducing the likelihood of preempting them.

To address these problems, in this chapter, we propose a QoS- and contention-aware scheduling policy in IGG level of InterGrid. Based on the taxonomy presented in Chapter 2, the solution proposed in this chapter is a meta-scheduling level solution for origin-initiated resource contention. This scheduling policy is comprised of two parts.

The first part, called workload allocation policy, determines the fraction of external requests that should be allocated to each RP (e.g., a Cluster) in a way that the number of VM preemptions is minimised. The proposed policy is based on the stochastic analysis of routing in parallel, non-observable queues. Moreover, this policy is a knowledge-free (i.e., it is not dependent on the availability information of the RPs). Thus, this policy does not impose any overhead on the system. However, it does not decide the RP that each single external request should be dispatched upon arrival. In other words, dispatching of the external requests to RPs is random.

Therefore, in the second part, called dispatch policy, we propose a policy to determine the RP to which each request should be allocated. The dispatch policy has the awareness of request types and aims to reduce the likelihood of contention (preemption) on valuable requests. In summary, this chapter makes the following contributions:

- It provides an analytical queuing model for a Grid, based on the routing in parallel non-observable queues.
- It adapts the proposed analytical model to a preemption-aware workload allocation policy.
- It proposes a deterministic dispatch policy to give more priority to more valuable users and meet their QoS requirements.
- It presents an evaluation of the proposed policies under realistic workload models and considering performance metrics such as number of VM pre-

emptions, utilisation, and average weighted response time.

The existing contention management policy in IGG level is based on adaptive partitioning of the availability times between local and external requests in each RP. In the current policy there is a communication overhead between RPs and IGG for submission of availability information. In addition to that, some RPs may not be willing to share their availability information for security reasons. Finally, there is a possibility that the availability information is inaccurate which deteriorates the scheduling results. By contrast, our scheduling method is non-observable and does not rely on availability information of RPs.

4.2 Analytical Queuing Model

The queuing model that represents a gateway (IGG) along with several RPs is depicted in Figure. 4.1. We consider each RP as a non-dedicated Cluster (i.e., Cluster with shared resources between local and external requests). There are N Clusters where Cluster j receives requests from two independent sources. One source is a stream of local requests with arrival rate λ_j and the other source is a stream of external requests which are sent by IGG with arrival rate $\hat{\Lambda}_j$. IGG receives external requests from other peer IGGs [131] (G_1,..,G_{peer} in Figure 4.1). Therefore, external request arrival rate to IGG is $\Lambda = \bar{\Lambda}_1 + \bar{\Lambda}_2 + ... + \bar{\Lambda}_{peer}$ where *peer* indicates the number of IGGs that can potentially send external requests to IGG.

Local requests submitted to Cluster j must be executed on Cluster j unless the requested resources are occupied by another local request or by a non-preemptable external request (see Chapter 3). The first and second moments of service time of local requests in Cluster j are τ_j and μ_j, respectively. An external request can be allocated to any Cluster but it might be subject to future preemption. We consider θ_j and ω_j as the first and second moments of service time of external requests on Cluster j, respectively. For the sake of clarity, Table 4.1 provides the list of symbols we use in this chapter along with their meaning.

The analytical model aims at distributing the total original arrival rate of external requests (Λ) amongst the Clusters. In this situation, if we consider each Cluster as a single queue and IGG as a meta-scheduler that redirects each incoming external request to one of the Clusters, then the problem of scheduling external requests in IGG can be considered as a routing problem in distributed parallel queues [132].

Considering this situation, the goal of the scheduling in IGG is to schedule

Table 4.1: Description of symbols used in the queuing model.

Symbol	Description
N	Number of Clusters
M_j	Number of computing elements in Cluster j where $1 \leq j \leq N$
$\bar{\Lambda}_j$	Original arrival rate of external requests to Cluster j
$\hat{\Lambda}_j$	Arrival rate of external requests to Cluster j after load distribution
Λ	$= \sum_{i=1}^{peer} \bar{\Lambda}_i = \sum_{j=1}^{N} \hat{\Lambda}_j$
θ_j	Average service time of a external request on Cluster j
ω_j	Second moment of external requests service time on Cluster j
γ_j	$= \theta_j \cdot \hat{\Lambda}_j$
λ_j	Arrival rate of local requests on Cluster j
κ_j	Arrival rate of local requests plus external requests to Cluster j
τ_j	Average service time of local requests on Cluster j
μ_j	Second moment of local requests service time on Cluster j
ρ_j	$= \tau_j \cdot \lambda_j$
m_j	$= \frac{\hat{\Lambda}_j}{\kappa_j}\omega_j + \frac{\lambda_j}{\kappa_j}\mu_j$
u_j	Utilisation of Cluster j $(= \gamma_j + \rho_j)$
r_j	Average response time of local requests on Cluster j
η_j	Number of VM preemptions that happen in Cluster j
T	Average response time of all external requests
T_j	Average response time of external requests on Cluster j
$\bar{v}_j$	Average number of VMs required by external requests
$\bar{d}_j$	Average duration of external requests
s_{ij}	Processing speed (MIPS) of processing element i in Cluster j

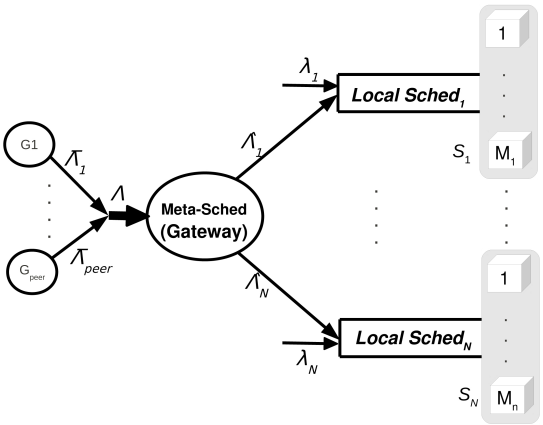


Figure 4.1: Queuing model for resource provisioning in a Grid with N RPs (Clusters).

the external requests amongst the Clusters in a way that minimises the overall number of VM preemptions in a Grid. Therefore, our primary objective function can be expressed as follows:

$$\min \sum_{j=1}^{N} \eta_j \tag{4.1}$$

To the best of our knowledge, there is no scheduling policy for such an environment with the goal of minimising number of VM preemptions. However, several research works have been undertaken in similar circumstances to minimise the average response time of external requests.

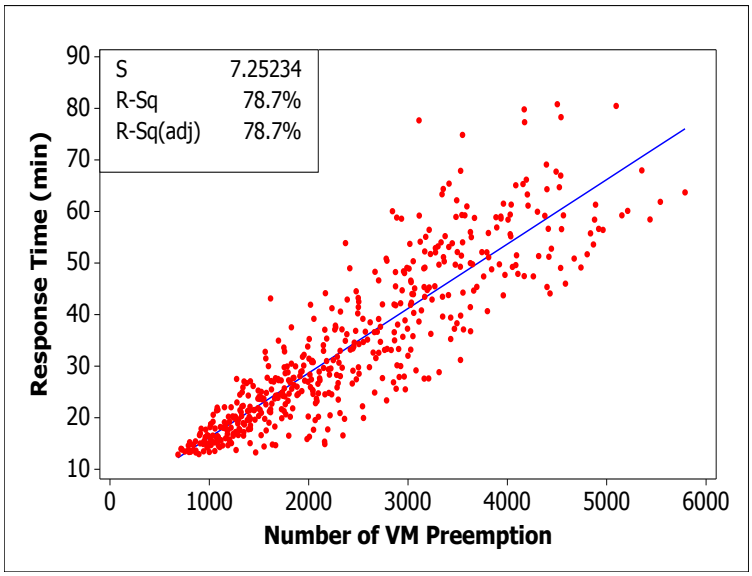


Figure 4.2: Regression between the number of VMs preempted and response time of the external requests.

Initial experiments intuitively suggest that there is an association between response time and number of VM preemptions in the Grid. The regression analysis with least squares method (depicted in Figure 4.2 and shown in Equation 4.2) demonstrates the positive correlation between the two factors. In Equation 4.2, R and η indicate the response time of external requests and number of VM preemptions, respectively.

$$R = 3.09 + 0.012\eta \tag{4.2}$$

Therefore, we expect that a reduction in the average response time has similar impact on the overall number of VM preemptions. Simulation results, which are discussed in Section 4.4.3, also confirm the correlation of response time and number of VM preemptions in the system. Details of the analysis are discussed over the next paragraphs.

For this purpose, we extend the approach developed by Li [5], which has been applied within a Cluster, for circumstances where there is a Grid system where some external requests are more valuable than others (i.e., they have different QoS levels).

Thus, we can define a new objective function that aims at minimising the average response time of the external requests (Equation 4.3):

$$T = \frac{1}{\Lambda} \sum_{j=1}^{N} \hat{\Lambda}_j \cdot T_j \tag{4.3}$$

Given the M/G/1 queue for each Cluster, and considering preemption of external

requests in favour of local requests, the response time of external requests in Cluster j (T_j) is defined based on Equation 4.4 [133].

$$T_j = \frac{1}{1-\rho_j}\left(\theta_j + \frac{\kappa_j m_j}{2(1-u_j)}\right) \tag{4.4}$$

The constraint for Equation 4.3 is:

$$\sum_{j=1}^{N} \hat{\Lambda}_j - \Lambda = 0 \tag{4.5}$$

The Lagrange multiplier method is applied to minimise Equation 4.3. We consider Equation 4.3 as $f(\hat{\Lambda}_j)$, Equation 4.5 as $g(\hat{\Lambda}_j) - c$, and z as the Lagrange multiplier. Then, the Lagrange function is defined as follows:

$$h(\hat{\Lambda}_j, z) = f(\hat{\Lambda}_j) + z\cdot\big(g(\hat{\Lambda}_j) - c\big) = \frac{1}{\Lambda}\sum_{j=1}^{N} \hat{\Lambda}_j\cdot T_j + z\cdot\big(\sum_{j=1}^{N} \hat{\Lambda}_j - \Lambda\big) \tag{4.6}$$

By solving the equations resulted from partial derivatives of all $\hat{\Lambda}_j (1 \leq j \leq N)$ and z, the input arrival rate of each Cluster is calculated based on Equation 4.7:

$$\hat{\Lambda}_j = \frac{(1-\rho_j)}{\theta_j} - \frac{1}{\theta_j}\sqrt{\frac{(1-\rho_j)(\omega_j(1-\rho_j)) + \theta_j\lambda_j\mu_j}{2\theta_j(1-\rho_j)z + (\omega_j - 2\theta_j^2)}} \tag{4.7}$$

Considering that $\Lambda = \hat{\Lambda}_1 + \hat{\Lambda}_1 + ... + \hat{\Lambda}_N$, then z can be calculated using the following Equation:

$$\sum_{j=1}^{N} \frac{1}{\theta_j}\sqrt{\frac{(1-\rho_j)(\omega_j(1-\rho_j)) + \theta_j\lambda_j\mu_j}{2\theta_j(1-\rho_j)z + (\omega_j - 2\theta_j^2)}} = \left(\sum_{j=1}^{N} \frac{(1-\rho_j)}{\theta_j}\right) - \Lambda \tag{4.8}$$

In fact, Equation 4.8 expresses the relation between different parameters of the system in which z is unknown. By solving Equation 4.8 for all Clusters and calculating z, Equation 4.7 can be solved. However, finding a generic closed form solution for z in Equation 4.8 is not possible [5]. Nonetheless, z can be found in the range [lb,ub] numerically. For this purpose, considering that $\hat{\Lambda}_j \geq 0$ and from Equation 4.7, we can infer that:

$$z \geq \frac{\lambda_j\mu_j}{2(1-\rho_j)^2} + \frac{\theta_j}{(1-\rho_j)} \tag{4.9}$$

Therefore, for all $1 \leq j \leq N$ the lower bound (lb) of the interval is:

$$lb = \max_{j=1}^{N} \left(\frac{\lambda_j \mu_j}{2(1-\rho_j)^2} + \frac{\theta_j}{(1-\rho_j)} \right) \tag{4.10}$$

If we define $\phi_j(z)$ according to Equation 4.11:

$$\phi_j(z) = \frac{1}{\theta_j} \sqrt{\frac{(1-\rho_j)(\omega_j(1-\rho_j)) + \theta_j \lambda_j \mu_j}{2\theta_j(1-\rho_j)z + (\omega_j - 2\theta_j^2)}} \tag{4.11}$$

and considering Equation 4.8, then we have:

$$\sum_{j=1}^{N} \phi_j(lb) \geq \left(\sum_{j=1}^{N} \frac{(1-\rho_j)}{\theta_j} \right) - \Lambda \tag{4.12}$$

The upper bound also can be determined based on Equation 4.13. ub can be reached by doubling lb up until the following condition is met.

$$\sum_{j=1}^{N} \phi_j(ub) \leq \left(\sum_{j=1}^{N} \frac{(1-\rho_j)}{\theta_j} \right) - \Lambda \tag{4.13}$$

If condition in Equation 4.12 is not met, then lb has to be decreased by removing Clusters which are heavily loaded. Load of the Cluster j is comprised of local requests that have been received and external requests which are already assigned to the Cluster. The load can be calculated as follows:

$$\psi_j = \frac{\lambda_j \mu_j}{2(1-\rho_j)^2} + \frac{\theta_j}{(1-\rho_j)} \tag{4.14}$$

For the sake of simplicity, in Equation 4.15 we assumed that $\psi_1 \leq \psi_2 ... \leq \psi_N$.

$$\sum_{j=1}^{k} \phi_j(\psi_k) \geq \left(\sum_{j=1}^{k} \frac{(1-\rho_j)}{\theta_j} \right) - \Lambda \tag{4.15}$$

It is worth mentioning that Clusters exceeding the value of k would not receive any external request from IGG (i.e., $\hat{\Lambda}_{k+1} = \hat{\Lambda}_{k+2} = ... = \hat{\Lambda}_N = 0$).

4.3 QoS- and Preemption-aware Scheduling

The proposed scheduling policy is comprised of two parts. The first part discusses how the analysis mentioned in previous section can be adapted as the workload

allocation policy for external requests in IGG. The second part is a dispatch policy that determines the sequence of dispatching external requests to different Clusters considering the type of external requests.

4.3.1 Workload Allocation Policy

The analysis provided in Section 4.2 was based on some widely accepted assumptions. Here, we state these assumptions and discuss if they are valid in the InterGrid scenario. In the analysis provided in Section 4.2 we assumed that:

- Each Cluster is an M/G/1 queue.
- All requests need one VM (i.e., they were sequential).
- Each queue runs in FCFS fashion.
- External requests are type-less (no superiority between external requests).

On the other hand, in our scenario we encounter parallel requests (requests that require more than one VM) that follow a general distribution. Additionally, we apply conservative backfilling [134] policy as the local scheduler of each Cluster. The reason of using conservative backfilling is that it increases the number of requests being served at each moment with respect to the FCFS policy [10]. Moreover, it is proved that conservative backfilling performs better in multi-Cluster environments compared to other scheduling policies [135]. Given M_j processing elements in Cluster j, and $\bar{v}_j$ the average number of VMs required by external requests, the number of simultaneous requests that are served within Cluster j is approximately $I_j \simeq M_j/\bar{v}_j$. We can infer that the queuing model of Cluster j is $G/G/I_j$.

However, in the analyses of Section 4.2 we applied the $M/G/1$ queuing model instead of $G/G/I_j$. This approximation was mainly because of mathematical tractability and it can be justified by the fact that if we consider the normalised response time in Equation 4.3, then the proportion of external workload to each Cluster remains unchanged. In other words, scaling up or down of the service times in the Clusters does not change the proportion of external requests allocated to each Cluster. In Section 4.4, we validate this approximation through extensive simulations in the context of workload allocation policy. Nonetheless, mathematical analysis of the $G/G/I_j$ queuing model in the Clusters can be considered as a future study.

Considering the above differences, we do not expect that the preemption-aware workload allocation policy performs optimally. In fact, we examine how

efficient the proposed analysis is in the InterGrid environment by relaxing these assumptions.

To adapt the analysis in a way that covers requests that need several VMs, we modify the service time of external requests on Cluster j (θ_j) and local requests on Cluster j (τ_j) in the following way:

$$\theta_j = \frac{\bar{v}_j \cdot \bar{d}_j}{\sum_{i=1}^{M_j} s_{ij}} \tag{4.16}$$

$$\tau_j = \frac{\bar{\zeta}_j \cdot \bar{\varepsilon}_j}{\sum_{i=1}^{M_j} s_{ij}} \tag{4.17}$$

where $\bar{\zeta}_j$ and $\bar{\varepsilon}_j$ show the average number of VMs needed and average duration of the local requests. Also, $\sum_{i=1}^{M_j} s_{ij}$ indicates the overall computing power offered by different processing elements within the Cluster j. Nonetheless, if the processing elements of a Cluster are homogeneous $\sum_{i=1}^{M_j} s_{ij}$ becomes $M_j \cdot s_{ij}$.

The second moment of the service time for both local and external requests are also accordingly changed. We use the coefficient of variance ($CV = StDev/Mean$) to obtain the modified second moment. Assuming that the CV is given, the second moment of service time for external and local requests on Cluster j is calculated according to Equation 4.18 and 4.19, respectively.

$$\omega_j = (\alpha_j \cdot \theta_j)^2 + \theta_j^2 \tag{4.18}$$

$$\mu_j = (\beta_j \cdot \tau_j)^2 + \tau_j^2 \tag{4.19}$$

where α_j and β_j show the CV of the external requests and CV of the local requests'service time on Cluster j.

The preemption-aware workload allocation policy (PAP) is presented in the form of pseudo-code in Algorithm 2. According to Algorithm 2, at first ψ is calculated for all Clusters. Then, in steps 4 to 10, to exclude the heavily loaded Clusters, they are sorted based on the ψ value in ascending order. Next, the value of k is increased until the condition defined in Equation 4.15 (step 7) is met. ub is found by starting from $2 \cdot lb$ and is doubled until the condition in step 13 is met. Steps 16-21 show the bisection algorithm mentioned in Section 4.2 for finding the proper value for z. Finally, in steps 22 and 23 the arrival rate to each Cluster is determined. Steps 24 and 25 guarantee that Clusters $k+1$ to N, which are heavily loaded, do not receive any external request.

Algorithm 2: Preemption-aware workload allocation Policy (PAP).

Input: $\bar{\Lambda}_j$,θ_j,ω_j,λ_j,τ_j,μ_j, for all $1 \leq j \leq N$.
Output: $(\hat{\Lambda}_j)$ load distribution of the external requests to different Clusters, for all $1 \leq j \leq N$.

1 **for** $j \leftarrow 1$ **to** N **do**
2 $\quad \psi_j = \frac{\lambda_j \mu_j}{2(1-\rho_j)^2} + \frac{\theta_j}{(1-\rho_j)}$;
3 `//Sort array` ψ `in ascending order`;
4 `Sort` (ψ);
5 $k \leftarrow 1$;
6 **while** $k < N$ **do**
7 $\quad$ **if** $\sum_{j=1}^{k} \phi_j(\psi_k) \geq \left(\sum_{j=1}^{k} \frac{(1-\rho_j)}{\theta_j} \right) - \Lambda$ **then**
8 $\quad\quad$ break;
9 $\quad$ **else**
10 $\quad\quad k \leftarrow k + 1$;
11 $lb \leftarrow \psi_k$;
12 $ub = 2 * lb$;
13 **while** $\sum_{j=1}^{k} \phi_j(ub) > \left(\sum_{j=1}^{k} \frac{(1-\rho_j)}{\theta_j} \right) - \Lambda$ **do**
14 $\quad ub = 2 * ub$;
15 `//`ϵ `is the expected precision`;
16 **while** $ub - lb > \epsilon$ **do**
17 $\quad z \leftarrow (lb + ub)/2$;
18 $\quad$ **if** $\sum_{j=1}^{k} \phi_j(z) \geq \left(\sum_{j=1}^{k} \frac{(1-\rho_j)}{\theta_j} \right) - \Lambda$ **then**
19 $\quad\quad lb \leftarrow z$;
20 $\quad$ **else**
21 $\quad\quad ub \leftarrow z$;
22 **for** $j \leftarrow 1$ **to** k **do**
23 $\quad \hat{\Lambda}_j = \frac{(1-\rho_j)}{\theta_j} - \frac{1}{\theta_j} \sqrt{\frac{(1-\rho_j)(\omega_j(1-\rho_j)) + \theta_j \lambda_j \mu_j}{2\theta_j(1-\rho_j)z + (\omega_j - 2\theta_j^2)}}$;
24 **for** $j \leftarrow k+1$ **to** N **do**
25 $\quad \hat{\Lambda}_j = 0$;

It is worth mentioning that, in practice, IGG can obtain the required parameters for this policy by analysing the Clusters'workload. Such parameters have been used in similar research works [136–138].

4.3.2 Dispatch Policy

The algorithm proposed in the previous subsection determines the routing probability to each Cluster (i.e., $\hat{\Lambda}_j/\Lambda$). However, it does not offer any deterministic sequence for dispatching each external request to Clusters (i.e., dispatching the external requests is memory-less). More importantly, as mentioned earlier, external requests are in different QoS levels and some external requests are more valuable. Hence, we would like to decrease the likelihood of contention (and therefore preemption) for more valuable requests to the minimum possible. We put this precedence in place through the dispatch policy.

In this part, we propose a policy that, firstly, reduces the number of VM contentions for more valuable external requests; Secondly, this policy makes a deterministic sequence for dispatching the external requests. It is worth noting that the dispatch policy uses the same routing probabilities that calculated for each Cluster using the workload allocation policy. The only difference is in the sequence of requests dispatched to each Cluster. For that purpose, we adapt the Billiard strategy [139] as the dispatch policy.

The Billiard strategy is a generalised form of Round Robin and considers the sequence of routing, which is called Billiard sequence. Suppose that a billiard ball bounces in an n-dimensional cube where each side and opposite side is assigned by an integer value in a range of $\{1, 2, ..., n\}$. Then, the billiard sequence is generated by a series of integer values that show the sides hit by the ball when shot. This sequence is deterministic, and is different from the sequence of probabilistic strategy, which is entirely random.

Hordikj [139], proposed a method to implement this strategy and generate the billiard sequence as follows:

$$j_s = \underset{\forall j}{min} \left\{ \frac{X_j + Y_j}{P_j} \right\} \tag{4.20}$$

where j_s is the *target* queue, and Y and X are vectors of integers with size n. Y_j keeps track the number of requests that have been sent to the queue j. X_j reflects which queue is fastest, and is set to one for the fastest queue and zero for all other queues [132]. Y_j is initialised to zero, and after the target queue is found, it is updated as $Y_{j_s} = Y_{j_s} + 1$. P_j is the fraction of external requests that

are sent to the queue j and is determined as the result of the workload allocation policy in Section 4.3.1.

It is worth mentioning that minimisation of the likelihood of preemption valuable requests depends on the scheduling policy in IGG (which is investigated in this chapter) as well as the local scheduling policy in each Cluster. The local scheduling policy we use in the Clusters has the awareness of the request types and preempts leases that belong to less valuable users (e.g., MOV policy in Chapter 3). Assuming such policy as the local scheduler of each Cluster, more valuable leases are preempted if and only if there is not (sufficient) leases of less valuable request types to be preempted. We can infer that the likelihood of contention for

Algorithm 3: Request Type Dispatch Policy (RTDP).

Input: P_j,θ_j for all $1 \leq j \leq N$.
Output: $SelectedCluster(j_s)$

1 $fastestCluster \leftarrow$ `findFastestCluster`(θ);
2 **foreach** *Cluster* j **do**
3 $\quad X_j \leftarrow 0$;
4 $\quad$ **foreach** *RequestType* i **do**
5 $\quad\quad P_j^i \leftarrow P_j *$ `GetProportion`(i);
6 $\quad\quad Y_j^i \leftarrow 0$;
7 $X_{fastestCluster} \leftarrow 1$;
8 **foreach** *external request received* **do**
9 $\quad i \leftarrow$ `GetRequestType`();
10 $\quad min \leftarrow MaxValue$;
11 $\quad$ **foreach** *Cluster* j **do**
12 $\quad\quad$ **if** $(P_j^i \neq 0)$ **then**
13 $\quad\quad\quad D = (X_j + Y_j^i)/P_j^i$;
14 $\quad\quad\quad$ **if** $(D < min)$ **then**
15 $\quad\quad\quad\quad min \leftarrow D$;
16 $\quad\quad\quad\quad tmpCluster \leftarrow j$;
17 $\quad Y_{tmpCluster}^i \leftarrow Y_{tmpCluster}^i + 1$;
18 $\quad j_s \leftarrow tmpCluster$;

valuable external requests would be low if a *mixture* of valuable and less valuable external requests are dispatched to each Cluster. Therefore, in the dispatch policy we keep track of number of external requests of each type that are dispatched to each Cluster. The pseudo-code developed for this purpose is presented in Algorithm 3.

In Algorithm 3, at first the fastest Cluster is found based on the average service time for external requests in each Cluster (step 1). We consider P_j^i as the probability of dispatching request type i to Cluster j. P_j^i is determined based

on P_j and the proportion of request type i in external requests (steps 4, 5). In step 7, we assign 1 to the fastest Cluster. Y_j^i expresses the number of external requests of type i that are dispatched to Cluster j and initially is zero (step 6). By receiving an external request, the value of the adapted billiard sequence for all Clusters are determined and a Cluster with minimum value is chosen (steps 9-16). Finally, Y^i is updated for the selected Cluster (step 17).

4.4 Performance Evaluation

4.4.1 Performance Metrics

User Satisfaction

As mentioned earlier, both resource owners and users benefit from fewer VM preemptions. From the resource owner perspective, less VM preemption leads to less overhead on the underlying system and improves the resource utilisation. From the external user perspective, fewer VM preemptions imply less resource contentions.

As one of the objectives of this chapter is giving more precedence to more valuable external users, we also investigate how distinct scheduling policies affect more valuable leases (i.e., deadline-constraint (DC) leases). To this end, for migratable leases we consider migration rate (percentage of migratable leases that are migrated) and for non-preemptable leases we consider the rejection rate (percentage of non-preemptable leases that are rejected).

Resource Utilisation

Time overhead due to VM preemptions leads to low resource utilisation. Thus, from the system owner perspective, we are interested to see how different scheduling policies affect the resource utilisation. Resource utilisation for one Grid system in InterGrid is defined as follows:

$$Utilisation = (1 - \frac{\sum_{j=1}^{N} overhead_j}{\sum_{j=1}^{N} computationTime_j}) \cdot 100 \tag{4.21}$$

where:

$$computationTime_j = \sum_{i=1}^{|L|} v(l_i) \cdot d(l_i) \tag{4.22}$$

where $|L|$ is the number of leases allocated in Cluster j, $v(l_i)$ is the number of VMs in lease l_i, $d(l_i)$ is the duration of lease l_i.

Average Weighted Response Time (AWRT)

Preemption-based scheduling policies are usually prone to long response time for best-effort (BE) requests (i.e., suspendable and cancellable requests). Therefore, in our study we are interested in the AWRT metric to see how the investigated scheduling policies affect the response time of the BE requests. Smaller values of AWRT indicate more (external) user satisfaction.

In fact, this metric measures the amount of time on average a BE lease should wait beyond its ready time to be completed. The AWRT in each Cluster is calculated based on Equation 4.23 [140].

$$AWRT_j = \frac{\sum_{l \in \Delta_j} v(l) \cdot d(l) \cdot (c_l - b_l)}{\sum_{l \in \Delta_j} v(l) \cdot d(l)} \tag{4.23}$$

where, Δ_j is the set of BE leases on Cluster j. c_l and b_l show completion time and ready time, $v(l)$ and $d(l)$ represent number of VMs and duration of lease l, respectively. Then, AWRT over all Clusters is defined as follows:

$$AWRT = \frac{\sum_{j=1}^{N} (M_j \cdot AWRT_j)}{\sum_{j=1}^{N} M_j} \tag{4.24}$$

4.4.2 Experimental Setup

To evaluate the performance of the scheduling policies, in GridSim [129], we consider a Grid with 3 Clusters with 64, 128, and 256 processing elements with different computing speeds ($s_1 = 2000, s_2 = 3000, s_3 = 2100$ MIPS). This means that in the experiments we assume computing speed homogeneity within each Cluster. This assumption helps us to focus more on the preemption aspect of resource provisioning. Moreover, considering that the resources are provisioned in the form of VMs, the assumption of homogeneous resources within the Clus-

ters is not far from reality [141]. It is worth noting that the analysis provided in Sections 4.2 and 4.3 are generic and do not consider homogeneity within Cluster nodes. The Cluster sizes are selected in accordance with the average demand of the current scientific applications [142]. LRMS of each Cluster employs conservative backfilling policy for scheduling. Clusters are interconnected using a 100 Mbps network bandwidth. We assume all processing elements of each Cluster as a single core CPU with one VM. The maximum number of VMs in the generated requests of each Cluster does not exceed the number of processing elements in that Cluster. We consider size of each VM as 1024 MB [14].

The overhead time imposed by preempting VMs varies based on the type of external leases involved in preemption [17] and is calculated based on the model provided in Chapter 3. In our experiments, suspension time (t_s) and resumption time (t_r) are considered as 161.0 and 126.1 seconds, respectively [17]. The time overhead for migrating a VM with similar configuration is 447.3 seconds.

Baseline Policies

We evaluate the proposed policies against other basic policies as well as recent policies which have been posed in other similar works [143]. These policies are described below:

- Round Robin (RR): In this policy IGG distributes external requests between Clusters in a round-robin fashion with a deterministic sequence. Formally, this policy is demonstrated as follows:

$$\hat{\Lambda}_j = \frac{\Lambda}{N} \tag{4.25}$$

- Least Rate First (LRF): In this policy the routing probability to each Cluster has inverse relation with arrival rate of local requests to that Cluster. Hence, IGG distributes the external requests with a random sequence between Clusters. In other words, Clusters that have larger arrival rate of local requests are assigned fewer external requests by IGG. Formal presentation of the policy is as follows:

$$\hat{\Lambda}_j = (1 - \frac{\lambda_j}{\sum_{j=1}^{N} \lambda_j}) \cdot \Lambda \tag{4.26}$$

- Biggest Cluster First (BCF): In this policy, the external requests assigned to Clusters with the probability proportional to their processing capability. This policy is commonly used in distributed systems [143]. This policy can

be formally described as follows:

$$\hat{\Lambda}_j = \left(\frac{\sum_{i=1}^{M_j} s_{ij}}{\sum_{j=1}^{N}\sum_{i=1}^{M_j} s_{ij}}\right)\cdot \Lambda \tag{4.27}$$

We have also implemented the workload allocation policy (PAP) with the following details:

- We assumed that in step 16 of Algorithm 2 the precision is 0.001 ($\epsilon = 0.001$). In fact, from the experiments we noticed that values greater than 0.001 do not change the results significantly.
- In Equations 4.18 and 4.19, to determine the second moment of service time for local and external requests, we assumed that in all Clusters $\alpha_j = 0.5$ and $\beta_j = 0.1$ (i.e., CV of service time for external requests is more than for local requests which implies that we expect more diversity in service time of external requests).
- To have a mixture of different external request types, in each workload there is a 25% of each external request type which are distributed uniformly over the generated requests.

We have implemented two dispatch policies for PAP. The first one in entirely random (PAP-RND in the experiments) and the other one which is the one described in Algorithm 3 (PAP-RTDP in the experiments).

Workload Model

In the experiments conducted, the DAS-2 workload model [33] has been configured to generate two-day-long workload of parallel requests. This workload model is based on the DAS-2 multi-Cluster Grid in the Netherlands.

We intend to study the behaviour of different policies when they face workloads with different characteristics. For this purpose, we change the specifications of external and local requests. Particularly, we study situations where:

- External requests have different number of VMs: In this case for external requests, we keep average *duration=420* seconds (similar to DAS-2 [33]), average *arrival rate=0.15*; and average *local request arrival rate=0.12*. In fact, local request arrival rate should not be too low (in this case few preemptions occur) and should not be too high (in this case there is no room for external requests). However, external request arrival rate should be higher than local arrival rate.

- The duration of external requests varies: In this case for external requests, we keep average *number of VMs=5* (similar to DAS-2 [33]), average *arrival rate=0.15*, and average *local request arrival rate=0.12*.

- The external requests'arrival rate varies: In this case for external requests, we keep average *number of VMs=5*, and average *duration=420* seconds, and average *local request arrival rate=0.12*.

- The local requests'arrival rate varies: In this case for external requests, we keep average *number of VMs=5*, average *duration=420* seconds, and average *request arrival rate=0.15*.

More details about the generated workloads are mentioned in Table 4.2. To generate these workloads, we modify parameters of DAS-2 model. As it is shown in Table 4.2, the distribution of local requests in each Cluster and also the distribution of external requests arriving to IGG are independent from each other. Based on the workload characterisation [33], the inter-arrival rate, request size,

Table 4.2: Input parameters for the workload model (C:Cluster).

Input Parameter	**Distribution**	**Values**	**Site**
No. of VMs	Log-uniform	$(l = 0.8, 2.5 \leq m \leq 3.5, h = 6, q = 0.9)$	Grid
		$(l = 0.8, m = 2.5, h = 6, q = 0.9)$	C64
		$(l = 0.8, m = 3.5, h = 7, q = 0.9)$	C128
		$(l = 0.8, m = 4.5, h = 8, q = 0.9)$	C256
Request Duration	Log-normal	$(3.0 \leq a \leq 5.4, b = 1.7)$	Grid
		$(a = 5.0, b = 1.7)$	C64
		$(a = 5.35, b = 1.7)$	C128
		$(a = 5.5, b = 1.7)$	C256
Inter-arrival Rate External Requests	Weibull	$(3.8 \leq \alpha \leq 7.0, \beta = 0.5)$	Grid
		$(\alpha = 2.0, \beta = 0.35)$	C64
		$(\alpha = 1.6, \beta = 0.35)$	C128
		$(\alpha = 1.2, \beta = 0.35)$	C256
Average Inter-arrival Rate Local Requests	Weibull	$(\alpha = 7.0, \beta = 1.1)$	Grid
		$(0.1 \leq \alpha \leq 7.0, \beta = 0.35)$	C64
		$(0.08 \leq \alpha \leq 6.0, \beta = 0.35)$	C128
		$(0.06 \leq \alpha \leq 4.5, \beta = 0.35)$	C256
P_{one}	N/A	0.2	Grid
		0.3	All Clusters
P_{pow2}	N/A	0.5	Grid
		0.6	All Clusters

and request duration follow Weibull, two-stage Log-uniform, and Log-normal distributions, respectively. These distributions with their parameters are listed in Table 4.2.

P_{one} and P_{pow2} are probabilities of request with one VM and power of two VMs in the workload, respectively. Hence, the mean number of VMs required by requests is given as follows:

$$\bar{v}_j = P_{one} + 2^{\lceil r \rceil}(P_{pow2}) + 2^r\,(1 - (P_{one} + P_{pow2})) \tag{4.28}$$

where r is the mean value of the two-stage uniform distribution with parameters (l, m, h, q) as listed in Table 4.2 and can be found as follows:

$$r = \frac{ql + m + (1 - q)h}{2} \tag{4.29}$$

Additionally, the mean request duration is the mean value of the Log-normal distribution with parameters (a, b) which is given by:

$$\overline{d}_j = e^{a+\frac{b^2}{2}} \tag{4.30}$$

Therefore, we are able to calculate the mean request size in Equations 4.16 and 4.17.

Each experiment is carried out on each of these workloads separately. For the sake of accuracy, each experiment is carried out 100 times by using different workloads and the average of the results is reported. In all the reported results the CV is less than 0.01. The results of the experiments are investigated from practical and statistical perspectives. In statistical analyses, we applied Two-way ANOVA and T-student tests. In doing these tests, we have ensured the normal distribution of the underlying data and the equity of variance.

4.4.3 Experimental Results

Number of VM Preemptions

The primary objective in this chapter is to express the impact of scheduling policies on the resource contention, which is measured by the number of VMs preempted within a Grid of InterGrid. Therefore, in this experiment we report the number of VMs preempted by applying different scheduling policies.

As we can see in all sub-figures of Figure 4.3, the number of VMs preempted rises by increasing the average number of VMs (Figure 4.3(a)), duration (Figure 4.3(b)), arrival rate of external requests (Figure 4.3(c)), and arrival rate of local requests (Figure 4.3(d)). In all of them PAP-RTDP statistically and practically significantly outperforms other policies (two-way ANOVA results in P-value<0.001 in all the cases).

The result of a T-test analysis between PAP-RTDP and PAP-RND in Figure 4.3(a) represents a significant difference. 95% confidence interval (CI) of the average difference between these policies is (2737.97, 3896.95) where P-value<0.001. Moreover, the 95% CI of the average difference between PAP-RND and LRF-RND

is (168.2, 1561.1) (P-value=0.02). This indicates that PAP-RND significantly outperforms other policies.

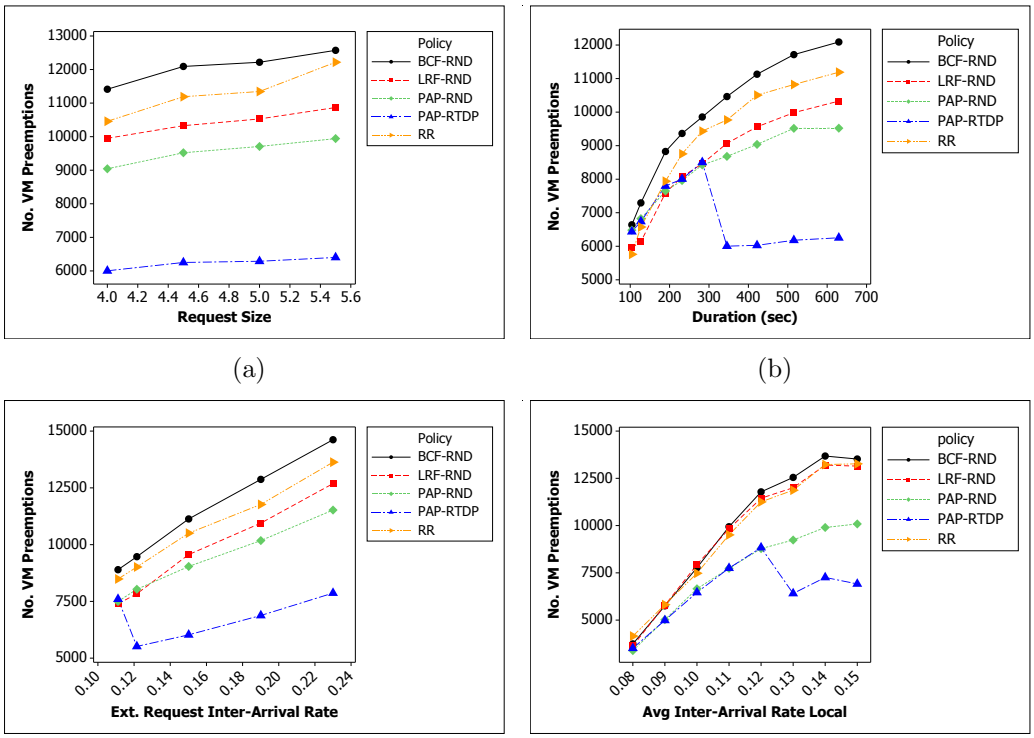


Figure 4.3: Number of VMs preempted by applying different policies. The experiment is carried out by modification of (a) the average number of VMs, (b) the average duration, (c) the arrival rate of external requests, and (d) the arrival rate of local requests.

In Figure 4.3(b), we witness a sharp decrease in PAP-RTDP when the average run time is more than 300 seconds. 95% CI of the average difference between PAP-RTDP and PAP-RND is (342, 2354.6) using T-test (P-value=0.012). Normally, as average duration increases, less free space is available, therefore, the arriving local requests result in more preemptions. Similar issue takes place in Figure 4.3(c) and 95% CI of the average difference between PAP-RTDP and PAP-RND using T-test is (380.5, 4570) and P-value=0.02 This means around 60% improvement over LRF-RND for durations more than 300 seconds. In fact, in the case of PAP-RTDP, better sequencing of the external requests resulted in balanced allocation, which leads to fewer preemptions and contentions.

Figures 4.3(c) and 4.3(d) reveal the efficacy of PAP-RND and PAP-RTDP, particularly where the arrival rate of external requests or the arrival rate of local requests increase. 95% CI of the average difference between these policies in Figure 4.3(c) is (380.5,4570) (P-value=0.02) and in Figure 4.3(d) for rates more than 0.12 is (469.12, 3826.45) (P-value=0.02).

As we noted, the difference between PAP-RTDP and PAP-RND is more remarkable in Figure 4.3(c) than Figure 4.3(d). However, in general, there is a larger difference between PAP (both RND and RTDP) and other policies in Figure 4.3(d) (95% CI of the average difference between PAP-RND and LRF-RND is (230.7, 4823.9) where P-value=0.03). We can conclude that workload allocation policy (PAP) has more impact where inter-arrival rate of local requests is high whereas dispatch policy has more influence where external requests'arrival rate is high.

Generally, the difference between PAP (specially PAP-RTDP) and other policies become more significant when there is more load in the system which shows the efficiency of PAP when the system is heavily loaded.

Resource Utilisation

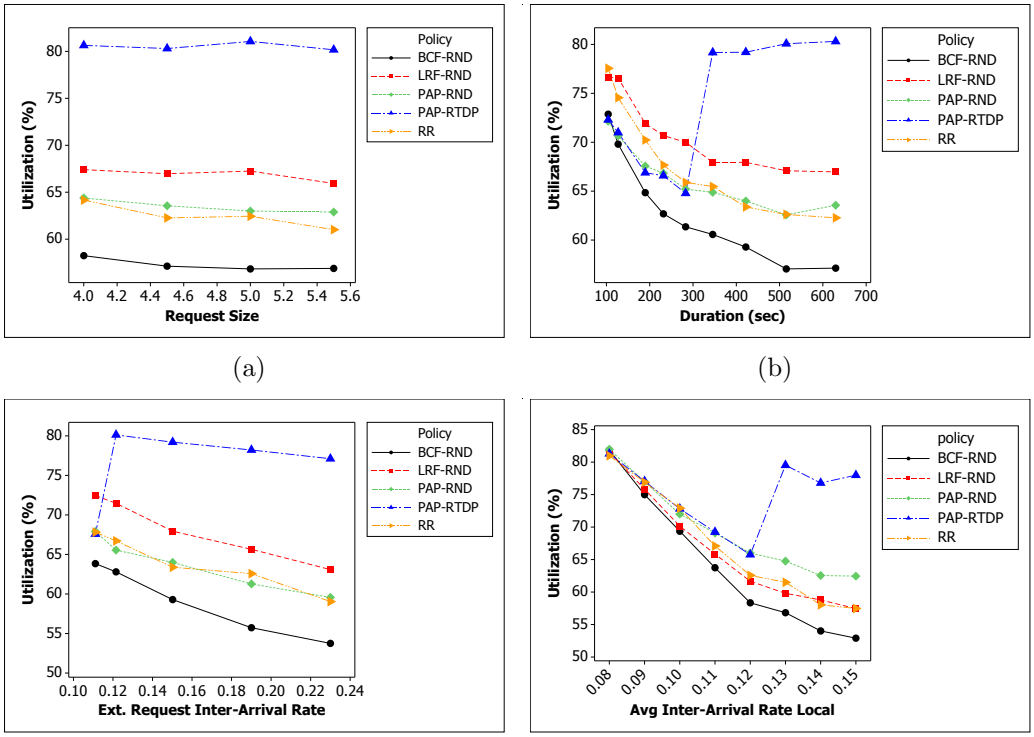
In this experiment, we explore the impact of preempting VMs on the resource utilisation as a system-centric metric.

In general, resource utilisation resulted from applying PAP-RTDP is drastically better than other policies as depicted in Figure 4.4. In Figure 4.4(a), 95% CI of the average difference of utilisation between PAP-RTDP and LRF-RND is (12.5, 14.7) (P-value<0.001) and the average difference between LRF-RND and PAP-RND is (2.2, 4.6) (P-value=0.001) using T-test.

However, the difference is more substantial when the average duration or arrival rate of external requests increases (Figures 4.4(b) and 4.4(c)). In Figure 4.4(b), 95% CI of the average difference between PAP-RTDP and LRF-RND for the durations more than 300 seconds using T-test is (7.3, 14.5) (P-value=0.002). Additionally, 95% CI of the average difference between LRF-RND and PAP-RND is (0.9, 7.4) with P-value=0.01. Also, in Figure 4.4(c), 95% CI of the average difference between LRF-RND and PAP-RTDP is (1.5, 15.1) (P-value=0.02). In Figures 4.4(b) and 4.4(c), the reason that LRF-RND leads to better utilisation comparing with PAP-RND is that LRF-RND rejects fewer requests and consequently utilises more resources than PAP-RND (see Figure 4.6(d) and 4.6(f)). Expectedly, PAP-RTDP performs better than other policies in Figure 4.4(d); 95% CI of the average difference between PAP-RTDP and PAP-RND for rates more than 0.12 is (11.3, 18.3)(P-value=0.001).

In Figure 4.4(b), we observe that PAP-RTDP results in better utilisation. The first reason is that PAP workload allocation policy is applied, which decreases the number of VM preemptions and consequently the overall overhead. The second reason is that PAP-RTDP is directed to prevent preempting migrat-

able leases (as a valuable lease) that impose significant overhead when compared with other lease types to migrate VMs. In other words, PAP-RTDP dispatches a balanced mixture of all request types to different Clusters. Therefore, the local scheduler can preempt BE leases that are less valuable and inherently impose less overhead to the system. In all other policies, in Figure 4.4(b), resource utilisation remains constant when external requests become longer (duration more than 300 seconds). The reason is that when requests are longer, the useful computation time dominates the overhead of VM preemptions. We can infer that VM preemption does not significantly affect resource utilisation when requests are long (more than 300 seconds).



(a) (b)

(c) (d)

Figure 4.4: Resource utilisation resulted from different policies. The experiment is carried out by modification of (a) the average number of VMs, (b) the average duration, (c) the arrival rate of external requests, and (d) the arrival rate of local requests.

Average Weighted Response Time (AWRT)

The results of this experiment for BE requests are shown in Figure 4.5 when different workload aspects vary. These results demonstrate that PAP-RND leads to minimum average weighted response time comparing to other policies. The reason that PAP-RTDP has longer response time than PAP-RND, is that the former leads to more preemptions on cancellable and suspendable leases. Consequently, the average response time of these requests increase.

Based on the results, we can conclude that PAP-RND results in better average response time for BE requests, which implies more satisfaction of BE users. More specifically, 95% CI of the average difference between PAP-RTDP and PAP-RND in Figure 4.5(a) is (3814.7, 4417) seconds (P-value<0.001). This difference in Figure 4.5(b) is (654.1, 4158) seconds (P-value=0.02) for requests longer than 300 seconds. However, there is not a statistically significant difference for requests shorter than 300 seconds (P-value=0.8). 95% CI of the average difference between PAP-RTDP and PAP-RND in Figure 4.5(c) is (673.8, 4753.5) seconds (P-value=0.02). Finally, 95% CI of the average difference between PAP-RTDP and PAP-RND in Figure 4.5(d) is (1516.4, 3784) seconds with P-value=0.005 for rates more than 0.12.

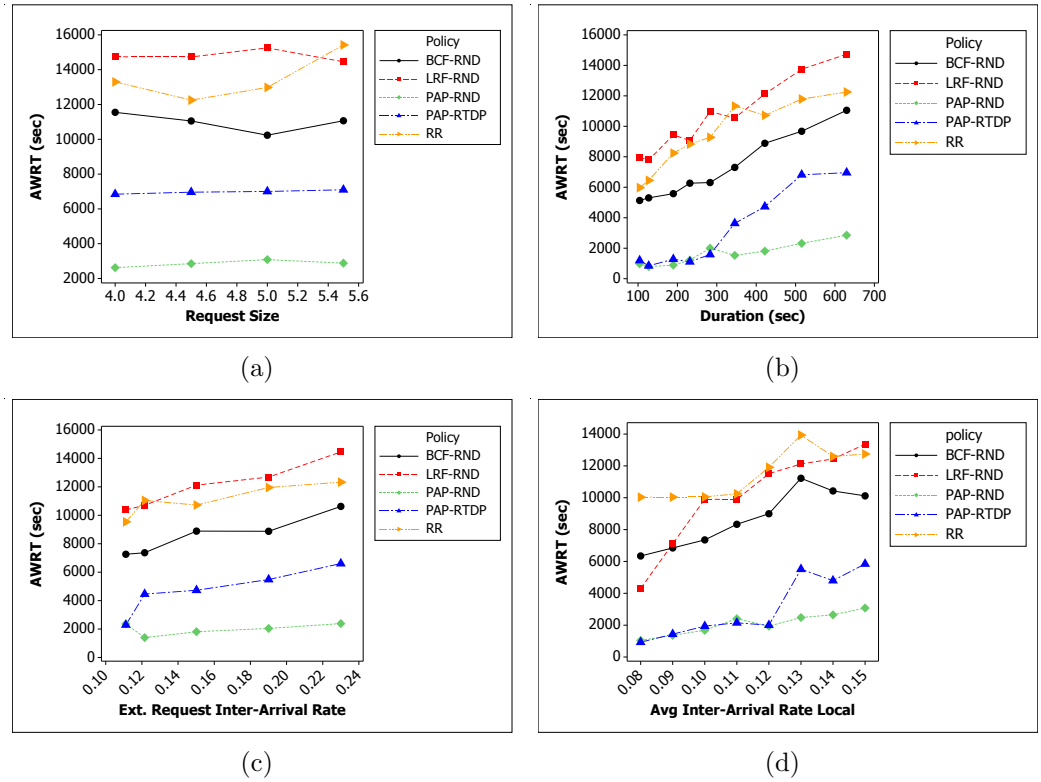


Figure 4.5: Average weighted response time resulted from different policies. The experiment is carried out by modification of (a) the average number of VMs, (b) the average duration, (c) the arrival rate of external requests, and (d) the arrival rate of local requests.

However, in all cases, PAP-RTDP performs significantly and practically better than other policies. More specifically, 95% CI of the average difference between PAP-RTDP and BCF-RND, in Figure 4.5(b), is (1863, 6456.2) seconds (P-value=0.002), in Figure 4.5(c) is (1665, 6115) seconds (P-value=0.004), and in Figure 4.5(d) is (3595.8, 7661) seconds with P-value<0.001. The reason that BCF-RND leads to lower AWRT than other policies (i.e., LRF-RND and RR), in spite of more preempted VMs (Figure 4.3), is that according to Figure 4.6, BCF-

RND results in more migrations and rejections comparing with other policies.

Another point in this experiment is that the AWRT does not change significantly by rising the average number of VMs in the external requests (Figure 4.5(a)) or their inter-arrival rate (Figure 4.5(c)). The reason is that in both cases, by increasing the average number of VMs of the external requests or their inter-arrival rate, more DC external requests and even more local requests are rejected. This makes more room for other requests to fit in. Hence, although the average number of VMs rises, AWRT does not increase.

Prioritising More Valuable External Users

In this experiment, we measure how different scheduling policies respect valuable users. We consider DC external requests (migratable and non-preemptable) as valuable users. For migratable requests we measure the number of times that VM migration happens (migration rate). For non-preemptive external requests we consider the rejection rate as the measurement criterion. The results of the experiments are illustrated in Figure 4.6.

This experiment expresses the efficacy of PAP-RTDP policy in migrating and rejecting fewer external requests. In all sub-figures of Figure 4.6, we notice that PAP-RTDP dispatch policy has substantially reduced the percentage of migrations and rejections. Details of the 95% CI of the average differences between PAP-RTDP and PAP-RND are presented in Table 4.3. According to Table 4.3, in almost all experiments PAP-RTDP leads to statistically and practically significant difference with PAP-RND. Except in Figure 4.6(h); Figures 4.6(c), and 4.6(d) where request duration is less than 300 seconds. P-values in these points are 0.8 and 0.7 respectively, which proves the null hypothesis (i.e., PAP-RTDP and PAP-RND are not statistically different).

In Figure 4.6(h), although PAP-RTDP is not statistically better than PAP-RND, we observe a marginal improvement in the rejection rate mainly for rates more than 0.12. We also witness a sharp decrease both in sub-figures 4.6(c) and 4.6(d) for requests that last more than 300 seconds. This is because the overall resource contention (number of preemptions) in that point has decreased (see Figure 4.3(b)).

In Figures 4.6(e) and 4.6(f) as the inter-arrival rate of external requests increases, we observe a decrease in the migration and rejection rates. In fact, more external requests raise the probability of having diverse leases at each time. This issue reduces the probability of migration and rejection. The issue is observed in Figures 4.6(a), 4.6(b) and thus we notice a slight decrease in rejection rate mainly

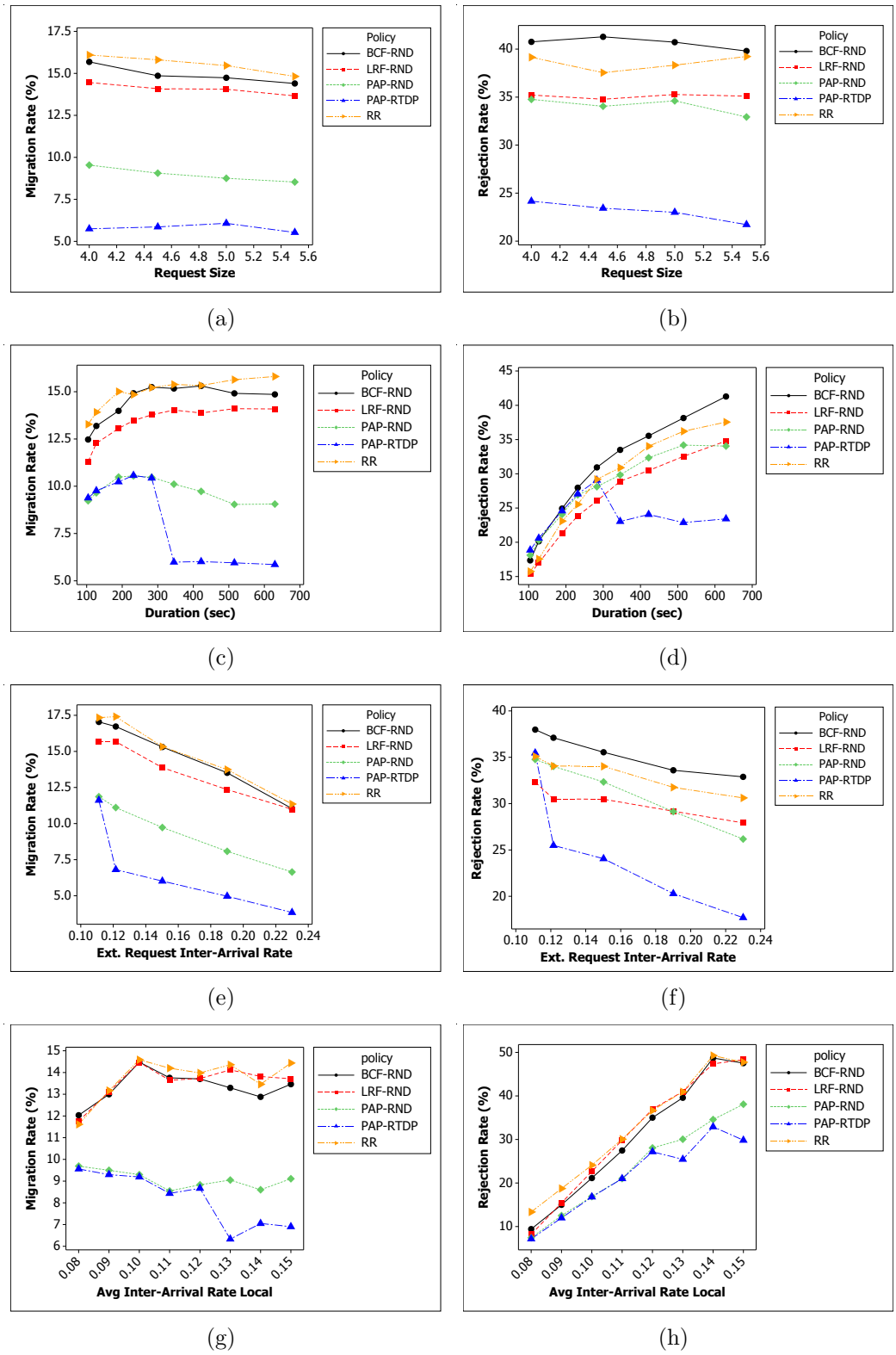


Figure 4.6: Respecting more valuable (deadline-constraint) users resulted from different policies. The experiment is carried out by modifying (a),(b) the average number of VMs, (c),(d) the average duration, (e),(f) the arrival rate of external requests, and (g),(h) the arrival rate of local requests.

in Figure 4.6(b).

Table 4.3: 95% confidence interval (CI) of the average differences between PAP-RTDP and PAP-RND related to Figure 4.6.

Figure	**95% CI**	**P-value**
4.6(a)	(2.4,3.8)	<0.001
4.6(b)	(9.3,12.7)	<0.001
4.6(c)	(2.6,4.3)	0.001
4.6(d)	(5.9,12.5)	0.003
4.6(e)	(0.14,7.8)	0.04
4.6(f)	(2.1,16)	0.02
4.6(g)	(0.02,3.2)	0.04
4.6(h)	Not statistically significant	0.2

4.5 Summary

In this chapter we explored how resource contention can be avoided through reduction of the number of preemptions in InterGrid. Particularly, we consider situations where some external requests are more valuable than others. For this purpose, we proposed a preemption-aware workload allocation policy (PAP) in IGG to proactively distribute external requests amongst RPs in a way that resource contention is reduced. Additionally, we investigated a dispatch policy that regulates dispatching of external requests in a way that the probability of contention for valuable requests is reduced. The proposed policies are knowledge-free and do not impose any communication overhead to the underlying system.

We compared the performance of the proposed policies with a variety of other policies. Experiment results indicate that PAP-RND and specifically PAP-RTDP significantly decreases the number of preemptions. We observed that PAP-RTDP leads to at least 60% improvement in VM preemptions comparing with other policies. This decrease in number of preemptions improves the utilisation of the resources and decreases average weighted response time of the external requests (by more than 50%). PAP-RTDP, particularly, is better for more valuable external requests and effectively leads to fewer preemptions for valuable external requests. Although PAP-RTDP, in general, preempts fewer VMs, PAP-RND results in better average response time for best-effort external requests. In fact, in case of PAP-RTDP, since most of the preempted leases are best-effort, it does not lead to minimum average response time. This indirectly represents the efficacy of PAP-RTDP for more valuable external requests. We also noticed that changing request size has little effect on the performance of scheduling policies.

This chapter provided scheduling policies in IGG to avoid and reduce resource contentions. However, resource contention is inevitable, particularly when there

is a surge in demand from local and external requests. Therefore, in the next chapter we provide an admission control policy that handles the side-effects of resource contention, in terms of long response time for external requests.

Chapter 5

Contention Management Using Admission Control in InterGrid

Resource owners are interested in accepting as many external requests as possible in order to maximise the utilisation. This increases the number of resource contentions and preemption of external requests which in turn leads to long and unpredictable response time for them. In these circumstances, the question that arises is: what is the maximum number of external requests that can be accepted by an RP in a way that they can be completed within specified response times. Admission control mechanisms can be employed to establish a predictable contention management system through limitation on the number of external requests accepted by an RP. In this chapter, we apply analytical queuing model to derive a preemption-aware admission control policy that addresses this question.

5.1 Introduction

In Chapter 3, we leveraged preemption of external requests in favour of local requests in InterGrid to assure that the local requests of an RP have priority access to the resources. However, preemption mechanism increases the waiting time and response time of external requests.

Long response time of external requests become more critical when an RP is over-subscribed to the external requests. In this situation, arrival of local requests causes more preemptions, and external requests are postponed in the scheduling queue. This increases the waiting time of external requests in the queue and consequently their average response time as shown in Figure 5.1.

In one hand, external users do not desire their requests being delayed because of resources'oversubscription. On the other hand, resource owners are interested

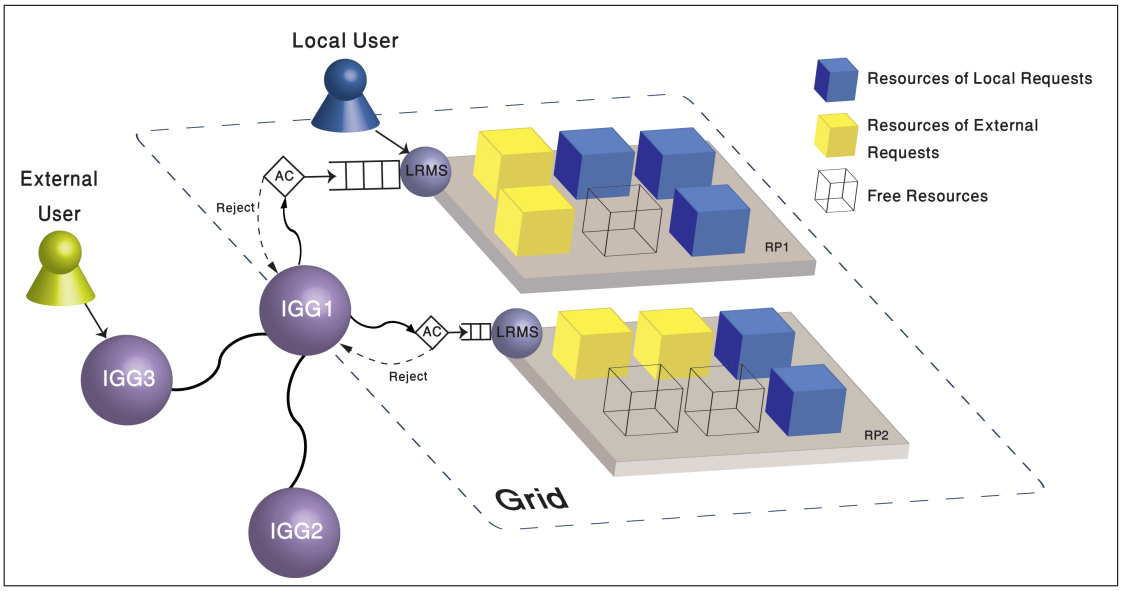


Figure 5.1: Illustration of queue for external requests in each RP of InterGrid.

to accept as many external requests as possible to maximise their resource utilisation. Therefore, the question that arises is: what is the maximum number of external requests an RP can accept in a way that long response time is avoided?

In general, there are several approaches in resource sharing environments to prevent long response times for requests. One approach is application of an *admission control* mechanism that prevents resources'oversubscription. As a result, the average response time decreases.

Sandholm et al. [52] investigate how admission control can fulfil users'QoS demands where there is a mixture of best-effort and QoS-constrained jobs. This admission control policy accepts a new request if current requests can still respect their QoS. Therefore, the overhead of feasibility test should be tolerated for each arriving request. However, we investigate an admission control policy that imposes less overhead on the system.

Gong et al. [7], provided a performance model to determine the run time of an external task in a single processor of a Network of Workstations (NOW) [7] where local and external requests coexist. Although the scenario is similar to the one we consider, the main difference is that they assume one external request at each moment (i.e., there is not any queue). Nonetheless, we focus on a scenario where a stream of external requests arrives to the RP.

Other research works on admission control either assumes non-prioritised environment, or did not consider the impact of VM preemption [42, 144, 145]. In this chapter we considers these issues and find out the ideal queue length of external requests (i.e., ideal number of external requests) in each RP. In a situation where the RP receives external requests beyond the ideal queue length,

the external requests are sent back to IGG (presented as "Reject" in Figure 5.1) for an alternative decision.

In this chapter, we propose a preemption-aware admission control policy within the LRMS of an RP. The objective of the policy is to maximise the number of accepted external requests and prevent long response times for external requests. We apply analytical queuing model to address this question.

5.2 Analytical Queuing Model

In this section, we describe an analytical model to determine the ideal queue length for external requests within the LRMS of each RP (we consider each RP as a Cluster). This section is followed by the proposal of an admission control policy, built upon the analytical model provided. Table 5.1 gives the list of symbols we use in this section along with their meanings.

Table 5.1: Description of symbols used in the queuing model.

Symbol	Description
$E(W_j)$	Expected waiting time of external requests in the LRM queue Cluster j
$E(T_j)$	Expected service time of external requests in Cluster j
$E(R_j)$	Expected response time of external requests in Cluster j
D	Waiting threshold for external requests
Λ_j	Arrival rate of external requests to Cluster j
μ_l^j	Service rate of local requests in Cluster j
μ_e^j	Service rate of external requests in Cluster j
ω	Mean duration of external requests
λ_j	Arrival rate of local requests to Cluster j
ρ_e^j	Λ_j/μ_e^j
ρ_l^j	λ_j/μ_l^j
α	Scale parameter in Gamma distribution
β	Shape parameter in Gamma distribution
θ_j	Coefficient of Variance (CV) for service time of local requests in Cluster j
e_i^j	ith running slice for an external request in Cluster j
l_i^j	Mean duration of local request i in Cluster j
$rate_l$	Low-urgency rate
u_l	Average deadline ratio for low-urgency requests
u_h	Average deadline ratio for high-urgency requests

The queuing model that represents a gateway (IGG in InterGrid) along with several non-dedicated Clusters (i.e., Clusters with shared resources between local and external requests) is depicted in Figure 5.2. According to this figure, Cluster j receives requests from two independent sources. One source is a stream of local requests with mean arrival rate λ_j and the other is a stream of external requests, which are sent by the gateway with mean arrival rate Λ_j.

The analytical model aims at determining the ideal queue length for external requests in each Cluster in a way that the response time of external requests is limited and the number of completed external requests is maximised. Our analysis is based on the following assumptions:

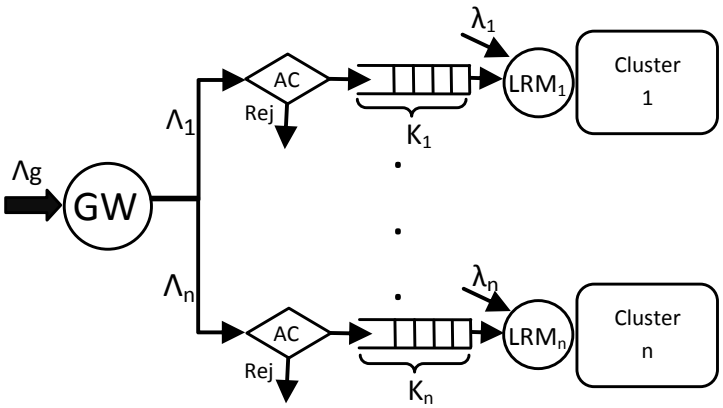


Figure 5.2: Queuing model for resource provisioning in a Grid with n Clusters.

- Requests can spread over all available resources in the Cluster (i.e., they are moldable).
- Local requests are deadline-constraint non-preemptable and must be processed when they are submitted. We assume an $M/G/1$ queue to model the service time of local requests.
- External requests are submitted to a queue in each Cluster that can be modelled as an $M/G/1/K$ queue model.

In this situation, the analysis goal is finding the suitable value of K_j for Cluster j in a way that the average response time for the requests in that queue will be less than an appointed threshold (called waiting threshold). Thus, our primary objective function is expressed as follows:

$$E(R_j) = E(W_j) + E(T_j) \leq D \tag{5.1}$$

where $E(R_j)$ is the expected response time; $E(W_j)$ and $E(T_j)$ are the expected waiting time and expected service time for external requests in Cluster j, respectively. D is the threshold for average response time of external requests. We call this factor as the *waiting threshold*. Over the next few paragraphs we discuss how $E(W_j)$, $E(T_j)$ are obtained for Cluster j.

If we suppose that an external request, with overall runtime ω, is subject to n preemptions before getting completed, then the service time (T) of the external request e can be formulated as follows:

$$T_j = e_1^j + l_1^j + e_2^j + l_2^j + ... + e_n^j + l_n^j + \epsilon \tag{5.2}$$

where l_i^j is the duration of the local request i and e_i^j is the ith running slice of

the external request e in Cluster j and we have $e_1^j + e_2^j + ... + e_n^j + \epsilon = \omega$. Also, ϵ is the last running slice of e. Given that the arrival rate of requests follows the Poisson distribution, we can conclude that e_i^j follows the exponential distribution and n follows Gamma distribution [133]. Therefore, we have $E(n) = \lambda_j \omega$ where λ_j is the arrival rate of local requests in Cluster j. Thus, $E(T_j)$ is determined based on Equation 5.3.

$$\begin{aligned} E(T_j) = E(E(T_j|n)) &= E(\omega + l_1^j + l_2^j + ... + l_n^j|n) \\ &= E(\omega + n\cdot E(l_1^j)) \\ &= \omega + \lambda_j \omega E(l_1^j) \end{aligned} \tag{5.3}$$

where $E(l_1^j) = 1/(\mu_l^j - \lambda_j)$ since it follows the $M/G/1$ queuing system. Hence, the expected service time and variance of service time for external requests ($E(T_j)$ and $V(T_j)$ respectively) are defined through Equations 5.4 and 5.5 [7]:

$$E(T_j) = \frac{\mu_l^j \cdot \omega}{\mu_l^j - \lambda_j} = \frac{\omega}{1 - \rho_l^j} \tag{5.4}$$

$$V(T_j) = \frac{\rho_l^j}{(1 - \rho_l^j)^3} \cdot \frac{\theta_j^2 + 1}{\mu_e^j} \cdot \omega \tag{5.5}$$

where θ_j is the coefficient of variance (CV) of service time for local requests; μ_e^j is the service rate of external requests; and ρ_l^j is the queue utilisation for local requests in Cluster j. According to Bose [146], the average waiting time of external requests in the $M/G/1/K$ queue is obtained based on Equation 5.6:

$$E(W_j) = \frac{1}{\Lambda_j} \sum_{k=0}^{K_j - 1} k\cdot P_{d,k}^j + \frac{K_j}{\Lambda_j}(P_{d,0}^j + \rho_e^j - 1) - E(T_j) \tag{5.6}$$

where, ρ_e^j is the queue utilisation for external requests and is calculated based on Equations 5.4 as follows:

$$\rho_e^j = \Lambda_j \cdot E(T_j) = \frac{\omega \cdot \Lambda_j}{1 - \rho_l^j} \tag{5.7}$$

Also in Equation 5.6, $P_{d,k}^j$ is the probability that a newly arriving external request observes k requests waiting in the queue of Cluster j. This is irrespective of whether or not the external request joins the queue. $P_{d,k}^j$ is obtained as follows [146]:

$$P_{d,k}^j = \frac{P_{\infty,k}^j}{\sum_{i=0}^{K_j - 1} P_{\infty,i}^j}, k = 0, 1, ..., K_j - 1 \tag{5.8}$$

Based on Equation 5.8, to obtain $P^j_{d,0}$, we need $P^j_{\infty,0}$ and $P^j_{\infty,k}$. $P^j_{\infty,0}$ is equivalent to the probability of zero length queue in an $M/G/1$ queue, which is: $P^j_{\infty,0} = 1 - \rho^j_e$ [133]. However, $P^j_{\infty,k}$ is obtained according to Equation 5.9 [146].

$$P^j_{\infty,k} = \frac{1}{\mu^j_e} \cdot \left(a_{k-1} \cdot P^j_{\infty,0} + \sum_{i=1}^{k-1} a_{K_j - i} \cdot P^j_{\infty,i}\right) \tag{5.9}$$

where a^j_k is defined as follows:

$$a^j_k = \int_0^\infty \frac{(t\lambda_j)^k}{k!} \cdot e^{-t\lambda_j} \cdot b_j(t) \cdot d_t \tag{5.10}$$

$b_j(t)$ in Equation 5.10 is the probability density function (PDF) of the service time for external requests in the Cluster j.

Gong et al. [7] showed that the service time of external requests in the presence of preemption in a Cluster follows a Gamma distribution. Therefore, we can apply the moment matching to acquire the parameters of the Gamma distribution (*scale*(α) and *shape*(β)). In this case, $\alpha\beta = E(T_j)$ and $\alpha^2\beta = V(T_j)$ and consequently α and β are obtained as follows:

$$\alpha_j = \frac{\rho^j_l(\theta^2_j + 1)}{\mu^j_e(1 - \rho^j_l)^2}, \beta_j = \frac{(1 - \rho^j_l)\mu^j_l \cdot \omega}{\rho^j_l(\theta^2_j + 1)} \tag{5.11}$$

Hence, $b_j(t)$ in Equation 5.10 can be calculated as follows:

$$b_j(t) = \frac{(t/\alpha)^{\beta-1} \cdot e^{-t/\alpha}}{\alpha \cdot \Gamma(\beta)} \tag{5.12}$$

where $\Gamma(\beta)$ is the Gamma function.

5.2.1 The Proposed Admission Control Policy

The analysis mentioned in the previous section can be adapted as the admission control policy for external requests within the LRMS. The positioning of this policy is demonstrated as "AC" (admission control) in Figure 5.2.

According to Equation 5.1, solving the queuing model requires determining the value of the waiting threshold (D), which is the upper bound of average response time that can be tolerated for external requests. The value of D is normally determined based on the amount of time each external request can possibly wait. It is also possible for the RP's administrator to appoint the value of the waiting threshold. In this case, the proposed model and the policy can

adaptively find the queue length for the external request in a way that the average response time of the external requests will be less than the waiting threshold.

In this dissertation, to express the time that each external request can wait, we define a deadline for each of them. Then, the waiting threshold is worked out based on the deadline of the external requests.

To generates the deadlines for external requests we have categorised external requests into *low-urgency* and *high-urgency* [39,147]. In practice, such categorisation can be based on the requests'criticality. In low-urgency requests the deadline is significantly greater than the runtime of the requests (i.e., $deadlineRatio = deadline/runtime$ is high). By contrast, in high-urgency requests the deadline ratio is low. Having the average deadline ratio for the low-urgency and high-urgency requests, the value of waiting threshold in Equation 5.1 (D) is:

$$D = (rate_l \cdot u_l \cdot \omega) + ((1 - rate_l) \cdot u_h \cdot \omega) \tag{5.13}$$

where $rate_l$ determines the percentage of external requests with low-urgency; also u_l and u_h are deadline ratios for low-urgency and high-urgency external requests, respectively.

Then, the preemption-aware admission control policy (PACP), which is built upon the analysis of Section 5.2, can be constructed. This policy is presented in the form of pseudo-code in Algorithm 4. In the beginning of the Algorithm (step 1), waiting threshold (D) is defined as described earlier. Next, in steps 4-10, in each iteration of the loop the queue length is increased by one until the average response time ($E(R)$), in step 9, exceeds D.

Algorithm 4: Preemption-aware Admission Control Policy (PACP) in Cluster j.

Input: Λ_j,θ_j,ω,λ_j,μ_e^j,μ_l^j,$rate_l$,u_l,u_h
Output: K_j (Queue length)

1 $D \leftarrow (rate_l * u_l * \omega) + ((1 - rate_l) * u_h * \omega)$;
2 $K_j \leftarrow 0$;
3 $ExpectedResponse_j \leftarrow 0$;
4 **while** $ExpectedResponse_j < D$ **do**
5 `/*calculating` $E(R)$ `for a queue with length` K_j `in Cluster` j`*/`
6 $\sigma \leftarrow 0$;
7 **for** $N_q^j \leftarrow 0$ **to** $K_j - 1$ **do**
8 $\sigma += N_q^j \cdot P^j_{d,N_q^j}$;
9 $ExpectedResponse_j \leftarrow \frac{1}{\Lambda_j} \cdot \sigma_j + \frac{K_j}{\Lambda_j}(P^j_{d,0} + \rho_e^j - 1)$;
10 $K_j \leftarrow K_j + 1$;

The output of the algorithm is K_j, which is the ideal number of external requests that can be admitted by Cluster j. Once K_j is obtained for Cluster j, the LRMS of the Cluster will not accept any external requests beyond K_j.

In practice, the LRMS in a Cluster can obtain the required parameters for PACP by analysing the Clusters'workload. Such parameters have been used in similar researches [136, 138].

Although we considered the Poisson arrival in the analysis, in the next section we examine how efficient the provided analysis and the proposed policy is under a real arrival model (i.e., non-Poisson).

5.3 Performance Evaluation

The performance evaluation of the policies are achieved in the context of InterGrid. For this purpose, we consider the workload received by an IGG through peering arrangements with other IGGs. IGG distributes the workloads amongst the RPs (Clusters) where each RP has its own local users. The admission control policy is embedded into the LRMS of each RP.

5.3.1 Performance Metrics

Violation Rate

This metric measures the percentage of external requests that waited beyond the waiting threshold. Users are interested in a policy that results in less violation rate. High values of this metric express less user satisfaction. The violation rate of external requests in a Grid is calculated based on Equation 5.14.

$$VR = \frac{(a \cdot v) + r}{a + r} \cdot 100 \tag{5.14}$$

where a and r are the number of accepted and rejected external requests in a Grid. v is the violation rate within the accepted external requests ($0 \leq v \leq 1$).

Completed External Requests

Admission control policies usually limit the number of requests accepted in a Cluster. This, however, conflicts with the resource owner's aim who benefits from accepting as many requests as possible. Therefore, we analyse how different admission control policies affect the number of completed external requests within

each Cluster and subsequently within a Grid.

5.3.2 Experimental Setup

We use GridSim [129], to evaluate the performance of the admission control policy. We consider a Grid of InterGrid with 3 RPs (Clusters) with 64, 128, and 256 processing elements and with different computing speeds ($s_1 = 2000, s_2 = 3000, s_3 = 2100$ MIPS) respectively. These sizes are in accordance with the average demand of current scientific high performance computing applications [142].

Each Cluster is managed by an LRMS with a conservative backfilling scheduler to improve the resource utilisation [135]. All processing elements of each Cluster have a single core CPU with one VM. Since we consider requests that contain moldable applications, the number of VMs required by a request adapts to the number of VMs available in the allocated Cluster and the duration (execution time) of the request changes accordingly.

The performance of our admission control policy also depends on the scheduling policy in the gateway (IGG) where incoming external requests are allocated to different Clusters (see Figure 5.1). To focus on the impact of admission control policies, we apply the round robin policy as the scheduling policy in IGG. Based on this policy, the arrival rate of external requests to Cluster j is: $\Lambda_j = \Lambda_g/n$ where Λ_g is the arrival rate to a Grid and n is the number of Clusters within that Grid.

Baseline Policies

We evaluate PACP against 3 other policies. Details of these policies are described below:

- Conservative Admission Control Policy (CACP): As a baseline policy, this policy admits as many requests as assigned by IGG. In fact, this policy favours resource owners since it does not reject any external request with the aim of maximising the number of completed external requests. From the queuing model perspective, this policy considers an $M/G/1/\infty$ queue within each Cluster, where the queue length is infinite.

- Aggressive Admission Control Policy (AACP): The other baseline policy considers the other extreme of spectrum where each Cluster accepts one external request at any time and tries to finish it within its threshold. We can argue that this policy favours accepted requests since it just tries to minimise violation rate of accepted requests.

- Rate-based Admission Control Policy (RACP): Sharifian et al. [148] proposed this policy which is the most similar to our policy in the area. In this policy, the queue length is determined based on the service rate for external requests and local request arrival rate in a Cluster (i.e., $N_q = \mu_e/\lambda$). We compare our proposed policy (PACP) with RACP to show its performance in comparison with recent works in the area.

Workload Model

In the experiments conducted, a workload model based on the Grid Workload Archive (GWA) [78] has been configured to generate a two-day-long workload of bag-of-tasks requests. This model is based on traces of 7 Grids over a year and is a good representative for Grid workloads.

For the sake of accuracy, each experiment is carried out 10 times by using different workloads. The results of the experiments are analyzed from practical and statistical perspectives. In the statistical analysis we applied T-student test and we have verified the *normality* of the underlying data as well as equity of variance. Also, we assured that the CV of all the reported results is less than 0.1.

In Table 5.2, different characteristics of the workload are described. Since the distribution of local requests and external requests are independent of each other in each Cluster, we mention them in distinct columns. The values of parameters in Table 5.2 are chosen based on realistic values collected and analyzed by Iosup et al. [78].

Table 5.2: Parameters of the workload model.

Input Parameter	**Distribution**	**external Requests**	**Local Requests**
Inter-arrival Time	Weibull	$(0.2 \leq \alpha_e \leq 3.2, \beta_e = 7.86)$	$(2 \leq \alpha_l \leq 10, \beta_l = 7.86)$
No. of Tasks	Weibull	$(a_e = 1.76, b_e = 2.11)$	$(a_l = 1.76, b_l = 2.11)$
Task Duration	Normal	$(60 \leq M_e \leq 110, \sigma_e = 6.1)$	$(60 \leq M_l \leq 110, \sigma_l = 6.1)$

In each experiment, we change one characteristic of the workload, while other characteristics are unchanged as follows:

- Arrival rate of local requests varies through changing α_l (we term it *local scale* which stands for the scale parameter in the Weibull distribution). For external requests, we keep $M_e = 90$ seconds and α_e=1.7 (called *external scale*). For local requests we keep $M_l = 90$ seconds.

- Task duration of local requests varies: We keep $\alpha_e = 1.7$, $M_e = 90$ seconds, and $\alpha_l = 4$.

- Arrival rate of external requests varies: We keep $\alpha_l = 4$, $M_l = 90$ seconds, and $M_e = 90$ seconds.
- Task duration of external requests varies: We keep $\alpha_l = 4$, $M_l = 90$, and $\alpha_e = 1.7$.

There are also some points on the value of parameters in the workloads:

- In Table 5.2, by increasing α_l and α_e the inter-arrival time increases (i.e., requests arrive less often). Therefore, as we expect that external requests arrive more frequently, we assign lower values of α to them.
- Mean duration of external tasks (ω), which is needed in Algorithm 4, is $\omega = meanNumberofTasks * meanTaskDuration$.
- To be realistic, the *local* workload assigned to each Cluster is proportional to the Cluster capacity (i.e., bigger Clusters receive more and bigger local requests). In fact, the values mentioned in Table 5.2 are the average characteristics of the *local* workload on the Cluster with 128 processing elements. On the Cluster with 64 processing elements, the mean task duration is decreased by 1 and the scale parameter (α_l) is increased by 1. In the Cluster with 256 processing elements, the mean task duration is increased by 1 and the α_l parameter is decreased by 1.

The GWA workload model does not have deadline for requests. Thus, similar to [39, 147], we synthetically assign deadlines to *low-urgency* and *high-urgency* external requests. The deadline ratio is distributed normally within each class of the requests. In our experiments, we consider the deadline ratio for low-urgency as: $N(4, 1)$ and for high-urgency external requests as: $N(2, 1)$. We executed the experiments for different values of deadline ratio in low and high-urgency requests and we got similar results. Finally, the arrival sequences of high-urgency and low-urgency request classes are distributed uniformly throughout the workload.

5.3.3 Experimental Results

Violation Rate

In this experiment we report the violation rate for external requests when different admission control policies are applied.

As we can see in all sub-figures of Figure 5.3, PACP resulted in less violation rate when compared with other policies. Specifically, we observe a rise in the

violation rate as the average duration of tasks in the local and external requests increases (Figures 5.3(a) and 5.3(c)). In Figure 5.3(b) and 5.3(d) we notice that the violation rate in all policies decreases when the inter-arrival time of local and external requests increase. In fact, in Figure 5.3(b), when the inter-arrival time of local requests increases, fewer preemptions occur for external requests and thus the violation rate decreases.

On the other hand, in Figure 5.3(d), as the external requests become less frequent, fewer external requests join the queue and existing external requests have more opportunities to complete before their deadline.

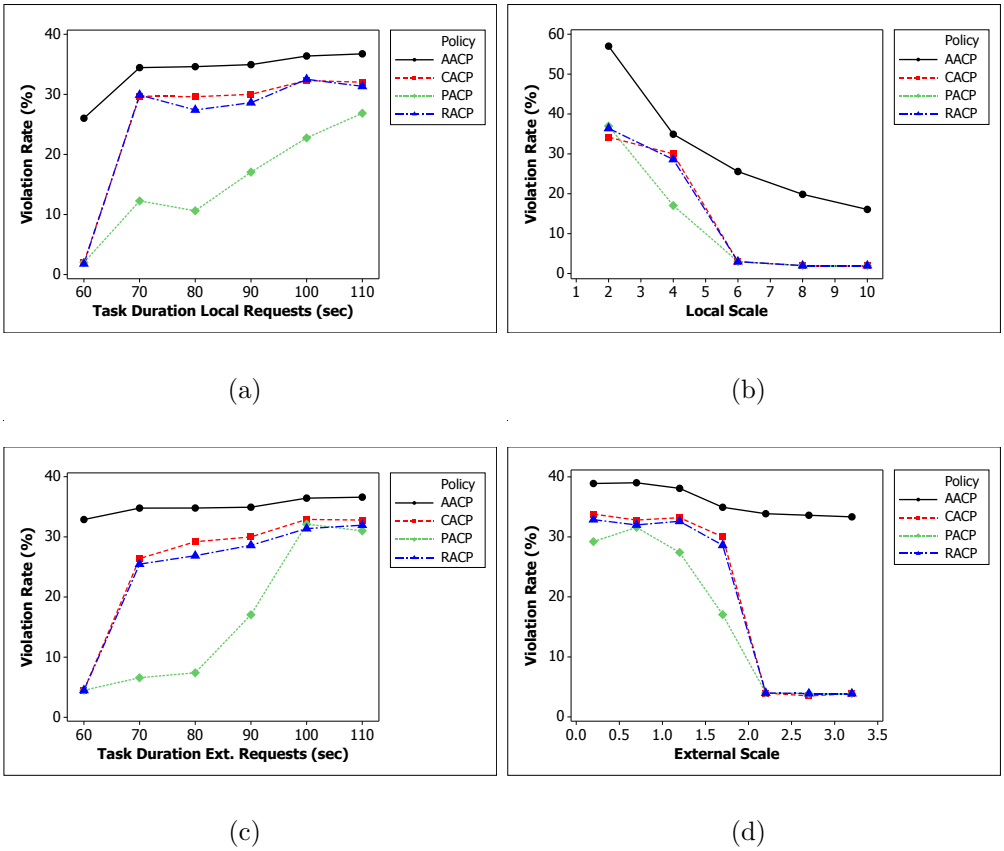


Figure 5.3: Violation rates resulted from different policies. The experiment is carried out by modifying (a) the average local task duration (M_l) and (b) the scale parameter (α_l) in local requests. In (c) and (d) for external requests with altering the mean task duration (M_e) and the scale parameter (α_e) as mentioned in Table 5.2.

In all cases, the difference between PACP and other policies is more significant when there is heavier load in the system, which shows the efficiency of PACP when the system is heavily loaded. The only exception is when task duration for external requests is long (more than 100 seconds in Figure 5.3(c)), which indicates that when the external requests are long, the queuing policies cannot affect the violation rate significantly. In the best case (in Figure 5.3(c), where the average

task duration is 80 seconds) we observe that PACP results in around 20% less violation rate when compared to RACP. In other points (between 70 and 90), the 95% confidence interval (CI) of the average difference between RACP and PACP is (14.79,18.56) where P-value<0.001.

This experiment also shows that although AACP accepts few external requests, its violation rate is the highest. This is because in Equation 5.14, the number of rejections is very high, therefore, the value of r dominates the result. RACP in these experiments performs similarly to the CACP. Particularly, in Figure 5.3(d), since inter-arrival time of external requests is increased, admission rate of RACP increases and approaches CACP. However, in Figure 5.3(a) and 5.3(c), as service rate of external requests decreases, a greater difference appears between RACP and CACP. In Figure 5.3(a), the 95% CI of the average difference between RACP and PACP is (14.12,17.86) and P-value<0.001.

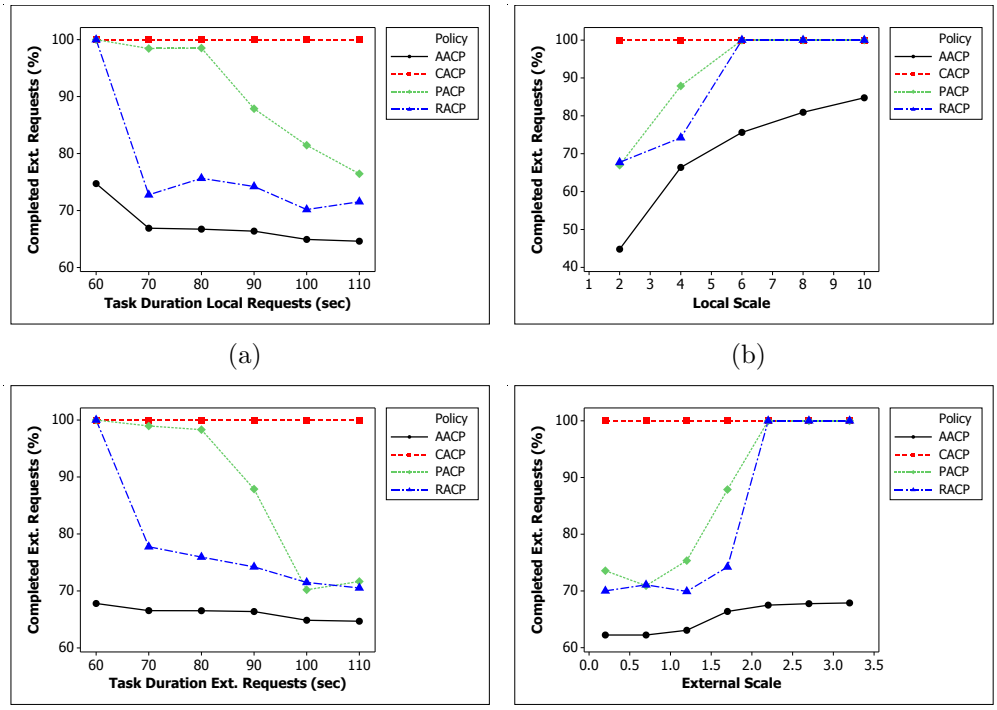
Completed External Requests

In this experiment, we report the percentage of external requests that are served. Different sub-figures in Figure 5.4 show how various policies affect the number of completed external requests from various aspects.

In general, we observe in all sub-figures of Figure 5.4 that AACP leads to the least number of completed external requests (because of an excessive number of rejections) whereas CACP results in the biggest number of completed external requests (always 100%) because it does not reject any of the external requests.

We also notice that PACP outperforms RACP in almost all cases. The superiority of PACP is particularly remarkable when the local/external requests are not long. According to Figures 5.4(a) and 5.4(c), the percentage of completed external requests reduces by increasing the task duration for both local and external requests. Since PACP adaptively decreases the queue length, the violation rate is minimised. Hence, the percentage of completed external requests decreases. Similarly, in RACP, by increasing task duration of local or external requests, service rate for external requests decreases and consequently, the number of completed external requests reduces. However, these figures show that PACP performs better in comparison with RACP.

The result of the 95% CI of the average difference between RACP and PACP in Figure 5.4(a) is (14.12,17.86) where P-value<0.001; In this figure, the maximum difference between the two policies is around 25% when task duration for local requests is 70 to 80 seconds. Moreover, the 95% CI of the average difference between RACP and PACP in Figure 5.4(c) is (17.09,21.3) and P-value<0.001.



(a) (b) (c) (d)

Figure 5.4: Percentage of completed external requests resulted from different policies. The experiment is performed by modifying: (a) the mean task duration for local requests (M_l) and (b) the scale parameter (α_l) for local requests. Also, in (c) and (d) with altering the mean task duration for external requests (M_e); and scale parameter (α_e) of inter-arrival time for external requests.

Figure 5.4(d) shows that PACP leads to completion of more external requests in comparison with RACP. We notice that, as the external requests become less frequent, PACP and RACP approach CACP. Finally, when the external scale parameter is more than 2.0, all the external requests are accepted.

5.4 Summary

In this chapter, we explored a predictable admission control policy that determines the ideal number of external requests within an RP of InterGrid. The ideal queue length is determined based on a proposed performance model in queuing theory. Then, we proposed a preemption-aware admission control policy (PACP) in the LRMS based on the performance model. We compared the performance of the proposed policy (PACP) with 3 other policies. Results of the experiments indicate that the PACP significantly decreases the violation rate for external requests (up to 20%). Additionally, PACP leads to completion of more external requests (up to 25%) in a two-day-long workload.

This chapter considers a situation where there is a surge in external requests

in RPs and provides an admission control policy to handle that. In the next chapter, we investigate a situation where resources in an RP operate at low utilisation and propose a contention-aware energy management mechanism for that.

Chapter 6

Contention-aware Energy Management in InterGrid

A considerable portion of the consumed energy in RPs is wasted because of idling resources. Indeed, one main motivation for the RPs to share their resources with external users in InterGrid is to avoid the resource wastage. Energy management mechanisms can be deployed in RPs to deactivate lightly loaded resources and decrease the wastage. However, decreasing the number of active resources increases the number of contentions and leads to long waiting time for external requests. The question we investigate in this chapter is how the energy consumption of an RP can be reduced without causing long waiting time for external requests. To answer this question we propose an adaptive energy management policy within the LRMS of an RP. The proposed policy adjusts the energy consumption of the RP based on the performance demands.

6.1 Introduction

In general, resource providers often function at low utilisation due to resource stranding and fragmentation [149]. Research studies revealed that the average utilisation of current resource providers is between 30% to 50% [67, 150] and the rest of resources are wasted. This wastage incurs a remarkable operational cost to resource providers, mainly because of their energy consumption. Particularly, as providers are rapidly growing in size and power, efficient management of their energy consumption is becoming more substantial.

This remarkable cost has recently raised the awareness about the energy consumption within resource providers. Thus, both industry and academia are seeking for energy efficient solutions in resource providers. Their overall goal is

reducing the energy consumption and adapting that to the user demands. Their proposed solutions encompass a spectrum from efficiency in cooling infrastructure and the hardware level to the resource management and algorithm design level.

Resource management solutions, in particular, were proven to be influential on reducing the energy consumption of resource providers [151]. Utilisation of Virtual Machine (VM) technology, as the resource provisioning unit, facilitates implementation of energy management policies in resource management systems [152]. VMs'capabilities in preemption and consolidation have been of special interest of researchers to implement energy management policies [67, 153–155]. VM consolidation [67] is a resource management functionality through which under-utilised resources are switched off and VMs running on them are migrated to other resources.

In InterGrid, one reason that RPs are motivated to share their resources with external requests is decreasing the resource wastage. However, there is still possibility that resources in an RP operate at low utilisation. This specifically can occur when there are many RPs within a Grid of InterGrid or the RP has many resources while there is not a huge demand for them. However, switching off resources and decreasing the number of active resources aggressively in an RP, raises the likelihood of resource contention and leads to long waiting time for external requests due to many preemptions. Arrangements are required in RPs to dynamically switch on resources in a way that their energy consumption is reduced, while external requests do not suffer from long waiting times.

Therefore, the research question that we investigate in this chapter is: How the energy consumption of an RP in InterGrid can be reduced in a way that external requests wait for a limited time and do not suffer from long waiting time?

We answer this question by proposing an adaptive energy management policy that dynamically increases and decreases the number of switched on resources in an RP. Specifically, the policy determines whether a new arriving local request should be served via preemption of other requests, or through reactivation of switched off resources. The proposed policy applies VM consolidation to save energy in circumstances that do not lead to long waiting time for external requests. The policy employs fuzzy logic in order to adaptively derive the appropriate decision.

Over the last few years, energy efficient resource management was extensively studied. Many of these studies employed VM consolidation for energy conservation. Another well-studied approach is the utilisation of dynamic voltage/frequency scaling (DVFS) technique.

For instance, pMapper [67] provides an energy-aware application placement platform in heterogeneous data centres that minimises energy and live migration costs while meeting performance guarantees.

In this chapter, we consider Haizea [13] as the local scheduler of the RP, as it is described in Chapter 7. We extend Haizea by adding energy management components to it and incorporated our proposed policy into that. Extensive experiments under realistic conditions indicate that the proposed policy significantly reduces energy consumption without any major waiting time for external requests.

6.2 Fuzzy Inference System

Fuzzy systems are based on fuzzy logic, which is a mathematical logic that deals with degrees of truth rather than true/false values as in the Boolean logic. In other words, fuzzy logic defines the concept of partial truth by values that range between completely true (1) and completely false (0). Hence linguistic values such as *low*, *rather low*, and *high* can be mathematically defined and employed for expressing rules and facts in a human-like way of thinking [156]. Nowadays, fuzzy logic is employed in numerous practical applications, such as control [157], prediction [158], and inference [159] systems.

A fuzzy inference system maps a set of given inputs to an output using the fuzzy logic [160]. This system utilises the concept of fuzzy sets in its mapping process. A fuzzy set is a set in which elements have partial degree of membership. A membership function is used to describe how each input value is mapped to a degree of membership. More importantly, fuzzy systems carry out the mapping through a set of condition statements which are in the general form of *If-then* rules. In fact, these rules capture the imprecise mode of reasoning and describe how human behave in similar situation in the system under consideration. An example of a rule in a two-input, single-output fuzzy inference system can be in the following form:

$$\begin{aligned} &\textbf{IF } \textit{risk} \textbf{ is } \textit{very low} \textbf{ AND } \textit{consumption} \textbf{ is } \textit{high} \\ &\textbf{THEN } \textit{decision} \textbf{ is } \textit{low consolidation} \end{aligned} \tag{6.1}$$

where *risk*, *consumption* and *decision* are linguistic variables that characterise the fuzzy sets of inputs and output. *very low*, *high*, and *low consolidation* are linguistic values.

The maximum number of fuzzy rules in a fuzzy system depends on the num-

ber of inputs to the fuzzy system as well as the number of fuzzy sets defined for each input. In fact, the maximum number of rules in a system is the product of the number of fuzzy sets of the input variables. For example, if a fuzzy system has $A < p, q, r >$ and $B < m, n >$ as inputs, then the maximum number of rules will be 6. Rules can be created from mapping each element in the Cartesian product of input fuzzy sets to the output fuzzy set. However, it is not compulsory that the rules cover the whole possible combinations of the input fuzzy sets.

The structure of a fuzzy inference system, which is shown in Figure 6.1, contains 4 main components as follows:

- *Fuzzifier*, which determines the membership degree of an input value for each input fuzzy set.
- *Fuzzy rule-base*, which contains a set of *If-then* rules.
- *Inference engine*, that carries out the inference operations based on the input values and the fuzzy rules.
- *Defuzzifier*, which converts the result of a fuzzy inference into a numeric value. In practice, systems require only numeric values to function, therefore, the defuzzifier is needed [161].

In addition to these 4 main elements, in some systems preprocessing and/or post-processing steps are also required to adapt the input and output values based on the system conditions. According to this structure, the fuzzy inference is carried

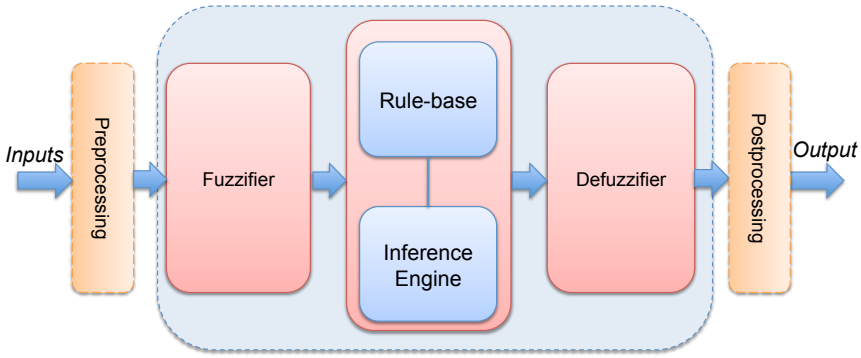


Figure 6.1: The structure of a fuzzy inference system.

out through the following steps:

1. For each input value, the fuzzifier obtains the membership value of that variable for each input fuzzy set.
2. Firing strength (weight) of each rule is determined based on the resulting value in step 1.

3. Based on the firing strength of each rule, the qualified output fuzzy set is determined.

4. The defuzzifier generates a numeric output by aggregating the qualified output fuzzy sets.

6.3 Proposed Energy Management Mechanism

In this section, we initially describe the proposed energy management policy. Then, we discuss the implementation details of energy-aware Haizea (i.e., the local scheduler of the RP) and how our proposed policy is incorporated into that.

6.3.1 Preemption-aware Energy Management Policy

The objective of the energy management policy is to reduce the energy consumption within an RP, while the local requests served within their requested time and external requests do not suffer from long waiting time [53].

To avoid long waiting time, we consider a situation where external requests also have a predictable and limited waiting time. For that purpose, the system administrator defines a maximum average waiting time (termed *waiting threshold*) for the external requests. The value of the waiting threshold shows the amount of time each external request can wait without suffering from long waiting time. It is also possible for the RP's administrator to appoint a value for waiting threshold based on his/her discretion. In any case, the proposed model should consider the threshold and schedule the external requests in a way that their average waiting time will be less than that.

Here, we assume waiting threshold to be α times longer than the average duration of external requests (i.e., $waiting\ threshold = \alpha \cdot |duration|$ and α is termed *waiting factor*). For example, the administrator can choose the waiting threshold to be 5 times of the average duration of the external requests.

In this situation, resource acquisition for an arriving local request can be carried out either via preemption of currently running external requests, or switching on resources. Also, in circumstances that the resource wastage is high, VM consolidation can be applied to reduce the energy consumption.

Specifically, preemption is applied when the risk of waiting threshold violation for external requests is low. By contrast, when the violation risk is high, the policy should switch on resources and offload the requests to them. As a result, external requests are not preempted and their average waiting time does not in-

crease. Finally, when the energy consumption is high and the average waiting time of external requests is sufficiently less than the waiting threshold, then the VM consolidation can be applied to save energy.

The risk of violating waiting threshold and the energy consumption are decisive variables for choosing an appropriate operation. These variables can be expressed using linguistic values such as low, medium, high, and etc. Considering the fuzzy logic power in modelling the linguistic variables in a system [162], we employ it to model the variables and infer the proper decision. Moreover, the fuzzy approach provides the adaptable solution that rapidly reacts to workload variations in the system.

The inputs of the proposed fuzzy engine are the violation risk and energy consumption. The output of the fuzzy engine is a value that drives the decision on how to allocate resources for an arriving local request. The output broadly can be switching on resources, preemption, consolidation, or a combination of these operations. We define violation risk of the waiting threshold as follows:

$$V = \frac{\tau}{\alpha \cdot E} \tag{6.2}$$

where α is the waiting factor, and E and τ are the average duration and average waiting time of external requests, respectively. E is calculated based on Equation 6.3.

$$E = \frac{\sum_{i=1}^{N} n_i \cdot d_i}{\sum_{i=1}^{N} n_i} \tag{6.3}$$

where N is the number of external requests waiting in the system, n_i is the number of requested resources, and d_i is the duration required by the external request i.

Also, τ in Equation 6.2 is defined based on Equation 6.4.

$$\tau = \frac{\sum_{i=1}^{N} n_i \cdot w_i}{\sum_{i=1}^{N} n_i} \tag{6.4}$$

where w_i is the waiting time of the external request i. Values more than one for violation risk ($V > 1$) indicates that external requests are waiting for more than the threshold.

The second input of the fuzzy inference system helps in deciding about preemption or switching on/off resources. For that purpose, we should know how the currently switched on resources are being utilised. Therefore, we consider the utilisation of the currently switched on resources (C) as the second input. C is defined according to Equation 6.5:

$$C = \frac{L}{\sum_{j=1}^{P} T_j} \tag{6.5}$$

where P is the number of switched on resources; T is the latest completion time of the current requests on resource j, and L is the total current load which is calculated based on Equation 6.6.

$$L = \sum_{i=1}^{N} d_i \cdot n_i \tag{6.6}$$

Values of C vary between $[0, 1]$. Low values for C shows that the switched on resources are operating at low utilisation. By contrast, values near to 1 indicate higher utilisation of the currently switched on resources. Based on the description, the fuzzy reasoning system can be expressed as follows:

$$\begin{aligned} &V \times C \rightarrow D \\ &C = \{VL, L, M, H, VH\} \\ &V = \{LR, MR, HR, VHR\} \\ &D = \{NP, QP, HP, 3QP, AP, LC, MC, HC\} \end{aligned} \tag{6.7}$$

where VL, L, M, H, VH indicate very low, low, medium, high, and very high fuzzy sets for C. LR, MR, HR, VHR are fuzzy sets of V and stand for very low risk, low risk, high risk, and very high risk, respectively.

D shows the output fuzzy sets of the fuzzy inference system which can range from NP, which means no preemption and resources should be switched on, QP, which means that quartile of the requested resources should be allocated through preemption and the rest has to be allocated via switching on resources. Similarly, HP and $3QP$ stand for half preemption and 3 quartile preemption. AP fuzzy set stands for all preemption and indicates that all resources should be allocated through preemption which implies that no additional resources should be switched on. Finally, LC, MC, and HC fuzzy sets indicate low, medium, and high consolidation of VMs, which help in the determination of the number

of resources that can be switched off.

Since there are 2 inputs with 4 and 5 fuzzy sets, the fuzzy rule-base has 20 rules. For instance, one rule in the fuzzy rule base is as follows:

$$\textbf{if } V \textbf{ is } M \textbf{ and } C \textbf{ is } H \textbf{ then } D \textbf{ is } HP \tag{6.8}$$

which means that if V is *medium* and C is *high*, then D is HP. This means that half of the requested resources have to be allocated via preemption and the other half through switching on resources. The fuzzy rule-base was formed based on our expectation from the system behaviour. Then, these rules were fine-tuned through extensive experiments and evaluating the outcomes in different conditions. The entire rule-base is listed in Table 6.1.

Table 6.1: List of rules used in the fuzzy rule-base.

Number	Violation Risk	Utilisation	Decision
1	LR	VL	HC
2	LR	L	HC
3	LR	M	MC
4	LR	H	LC
5	LR	VH	AP
6	MR	VL	MC
7	MR	L	LC
8	MR	M	LC
9	MR	H	HP
10	MR	VH	3QP
11	HR	VL	HP
12	HR	L	HP
13	HR	M	QP
14	HR	H	QP
15	HR	VH	NP
16	VHR	VL	QP
17	VHR	L	QP
18	VHR	M	NP
19	VHR	H	NP
20	VHR	VH	NP

The functionality of the fuzzy inference system is expressed in Equation 6.9:

$$f(x) = \frac{\sum_{r=1}^{R} \bar{y}^r \cdot \mu_C^r(x_1) \cdot \mu_V^r(x_2)}{\sum_{r=1}^{R} \mu_C^r(x_1) \cdot \mu_V^r(x_2)} \tag{6.9}$$

where r indicates a fuzzy rule and R is the total number of rules in the rule-base (i.e., $R = 20$); x_1 and x_2 are the current values of C and V, respectively, and $\mu_C^r(x_1)$ and $\mu_V^r(x_2)$ show the membership value of the x_1 and x_2 in the membership

function of rth rule. Finally, $\bar{y}^r$ expresses the center of fuzzy membership function fired by rth rule from the output fuzzy set. $f(x)$ covers values more than -1.

We used triangular membership function for all inputs and output variables, as shown in Figure 6.2. Also, we implemented the fuzzy system using a singleton fuzzifier, product inference engine, and center of gravity defuzzifier [162].

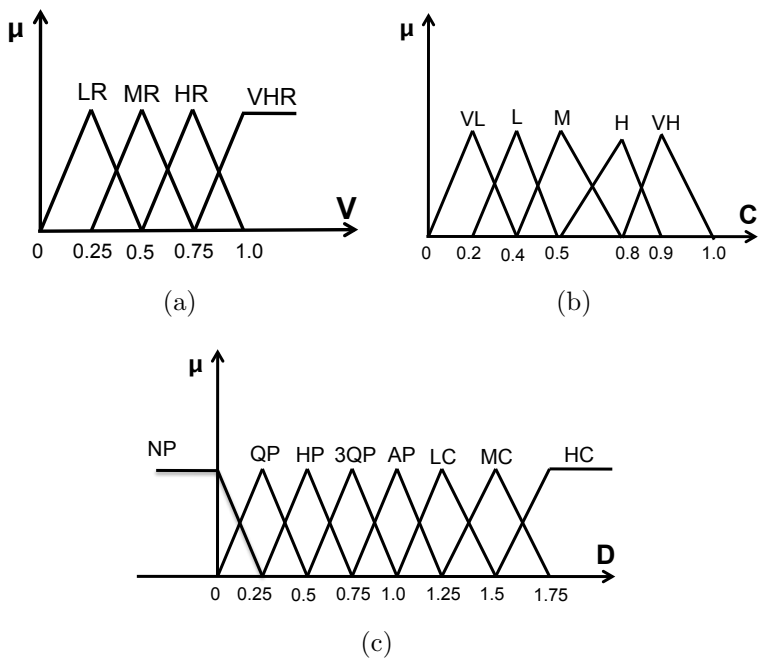


Figure 6.2: Fuzzy sets considered for inputs and output in the proposed system. (a) Violation risk (V), (b) Utilisation (C), and Decision (D).

It is worth noting that the proposed policy is not a scheduling policy. Indeed, it is the "energy management" component of the LRMS, which works closely with the scheduler but it is not the scheduler. The proposed policy determines how resources should be allocated for a new local (high priority) request. Then, a scheduling policy, such as backfilling, handles the scheduling of requests on the existing resources.

6.3.2 Energy-awareness in Haizea

Haizea [13] is an open source platform that can be used as the scheduling backend of a virtual infrastructure manager, such as OpenNebula [36], within an RP.

Haizea, by default, assumes that all resources are switched on and are ready to be utilised. To add energy-awareness to the Haizea scheduler, this assumption has to be relaxed. In the energy-aware Haizea, the assumption is that resources are switched off initially. Then, as the time passes and the demand increases, the resources are switched on. Accordingly, when there is not any scheduled

request on a resource, the resource is switched off. Adding these capabilities entails significant modification in the architecture of Haizea.

As a result of these modifications, Haizea lease scheduler is equipped with the following main functionalities:

- Switching on resources in an on-demand manner. Here, on-demand refers to situation that the number of switched on resources is not adequate to serve local requests. In our energy management policy, we have extended the on-demand switching on resources also to situation that the risk of violation is high.
- Switching off the resources when they are not required. This occurs when there is not any scheduled request on a resource.
- VM consolidation which takes place when some resources operate at low utilisation. In these circumstances, the VMs running on the target resources have to be rescheduled and re-allocated. Then, the resource can be deactivated. In the implementation, we apply VM consolidation on resources that have the fewest number of leases scheduled on them.
- The scheduler was also modified in a way that it just considers switched on resources at each moment. In other words, the scheduler is enabled to dynamically add and remove resources from the scheduling.

Apart from adding the major functionalities mentioned above, there are many other minor changes in the new structure. We uploaded the energy-aware version of the Haizea to our web site[1]. Interested readers should be able to understand the modifications clearly by downloading and reviewing the code and documentations. We developed our system in a pluggable way that enables other researchers to develop their own energy management policies.

6.3.3 Incorporating the Preemption-aware Energy Management Policy (PEMP) into the LRMS

After implementing the basic functionalities for energy-awareness in Haizea, the policy proposed in Subsection 6.3.1 can be implemented and incorporated into the Haizea. The pseudo code of the implemented policy is illustrated in Algorithm 5.

The algorithm is executed for each arriving local request and decides about the resource allocation for the request. Additionally, it runs periodically to avoid long waiting time for external requests or resource wastage in the RP.

[1] http://ww2.cs.mu.oz.au/~mohsena

The algorithm inputs are: the average waiting time (τ), the average duration (E), and the appointed waiting factor (α) for the external leases. The arriving local request that has to be scheduled (req) is the other input of the algorithm. The result of the algorithm is the proper operation that should be performed.

As we can see in the beginning of the algorithm, V and C are calculated based on Equations 6.2 and 6.5, respectively. In line 3, the fuzzy reasoning is invoked based on the values of C and V and the details discussed in Subsection 6.3.1. Then, from lines 4 to 18, the appropriate operation is performed via post-processing on the output of the fuzzy engine (f, where $f \geq -1$).

Algorithm 5: Preemption-aware Energy Management Policy (PEMP).

```
   Input: α,τ,P,E,L,req
1  V ← τ/(α· E);
2  C ← L/(P·T);
3  f ← FuzzyReasoning(V, C);
4  if f ≤ 1 then
5      if f < 0 then
6          Num ← getNumNodes(req);
7      else if f > 0 and f ≤ 1 then
8          Num ← getNumNodes(req) ∗ (1 − f);
9      SwitchOnNodes(Num);
10     Preempt(getNumNodes(req) − Num);
11 else
12     /*Consolidation*/
13     if f > 2 then
14         Num ← getNumNodes(req);
15     else Num ← getNumNodes(req) ∗ (f − 1);
16     minNode ← Required(req.strtTime);
17     if minNode ≤ (NumSwitchOnNodes − Num) then
18         Consolidate(Num);
```

The way resources are allocated to an arriving local request is driven by the value of f. $0 < f < 1$ shows the situation where resources should be provided via switching on resources. As f approaches 1, fewer resources should get switched on (line 8) and more resources should be allocated via preemption. Specifically, $f = 1$ does not lead to switching on any resource and all resources should be allocated via preemption. In lines 9 and 10, for a given request it is determined how many of the VMs should be allocated through switching on resources and how many should be provided through preemption of running VMs.

There are also some specific cases that we did not mention in the algorithm to keep it simple and readable, however, we considered them in our implementation.

For example, in line 9, there may not be adequate resources in the RP to be switched on. In this situation we switch on as many resources as we can and the rest of the resources are allocated through preempting VMs.

$f > 1$ shows the situation that the violation risk is low in a way that VM consolidation can be carried out. For consolidation (line 12 onwards), after deciding how many of the resources can be consolidated (lines 13 to 15), the algorithm must determine if switching off that many resources affect currently scheduled local leases or not. Therefore, it calculates the minimum number of resources required at that time (line 16). If switching off resources do not affect the local leases (line 17), then the consolidation is carried out (line 18). For consolidation, the algorithm uses a greedy approach and considers the resources that have minimum leases scheduled on them. Also, the algorithm assumes that in line 18 the function $Consolidate(Num)$ includes handling affected leases (i.e., preempting and rescheduling them) within itself.

6.4 Performance Evaluation

Performance evaluation is achieved using the energy-aware Haizea, which has been adopted as the local scheduler of the RPs. To be able to achieve extensive experiments and evaluate the performance of the system under various workload conditions, we conducted the experiments in the emulation mode of the Haizea.

6.4.1 Experimental Setup

To have a realistic evaluation, the experiments are carried out based on real traces from the Blue Horizon cluster [163] in San-Diego Supercomputer Center (SDSC). Therefore, we consider an RP with 512 single-CPU nodes, each having 1GB of memory, and 1Gbps bandwidth between them.

Conservative backfilling [10] is used as the scheduling policy of the RP. We also assume that each node can run one VM. However, our proposed policy encompasses multi-core systems where multiple VMs can exist on the same node.

Baseline Policies

We evaluate the proposed policy against 2 other policies, which are used as a benchmarks. Details of these policies are described below:

- *Greedy Energy Saver Policy (GESP):* In order to maximise energy conser-

vation, this policy switches on the minimum number of resources required for the local requests. Then, the external requests can be scheduled in the remaining time slots (we call it scheduling fragments) of local leases.

- *Starvation-aware Energy Saving Policy (SESP):* This policy favours external requests by trying to ensure that the waiting threshold is not violated. Therefore, whenever the violation risk is high ($V \geq 1$), the policy switches on resources based on the number of resources required by the request.

In the implementation of PEMP, we have considered waiting factor as 5 ($\alpha = 5$).

Workload Model

To have a combination of local and external requests, similar to Sotomayor et al. [13, 17], we extract 30 days of job submissions from the trace (5545 submissions). These requests are treated as external requests and then, an additional set of local requests are interleaved into the trace.

For the sake of accuracy, we evaluate the performance of different policies under various workload conditions. For that purpose, we keep the external requests fixed and generate 72 different workloads by varying local request characteristics as follows:

- ρ, the aggregate duration of all local requests within a workload, which is computed as a percentage of the total CPU hours in the whole workload. We investigate values of $\rho = 5\%, 10\%, 15\%, 20\%, 25\%, 30\%$. The reason that we do not explore larger values for ρ is that, in practice, the RPs'utilisation is between 30% to 50% [67, 150]. Considering that the trace's utilisation (external requests) is 34.8%, the overall utilisation (external and local) is between 39.8% and 64.8%.

- δ, the average duration of local requests. In the experiments we explore the values of $\delta = 1, 2, 3, 4$ hours which is similar to the trace's duration. For generation of the duration of the local requests, we select the duration randomly in the range of $\delta \pm 30$ minutes.

- θ, the number of nodes requested by each request. For this parameter we use 3 distinct ranges, namely, *small* (between 1 and 24), *medium* (between 25 and 48), and *large* (between 49 and 72). We choose the number of requested nodes for each request based on a uniform distribution.

- To realise the impact of RP size, we conducted the same experiments when

the number of nodes in the RP is different. Specifically, we consider situation that the RP has 144, 256, and 512 nodes.

Based on the above parameters, we can determine how many local requests are generated. Using the number of generated local requests in 30 days, we can find out the average arrival rate of the local requests in each day (λ). Then, the individual interval between two local requests is randomly selected in the range of ($\lambda - 1$ hour and $\lambda + 1$ hour).

To investigate the impact of each parameter, in each experiment, we modify one of the above parameters while keeping the rest constant. When we modify ρ, we keep $\delta = 3$ hours and $\theta = medium$. When δ is changed, we keep $\rho = 15\%$ and $\theta = medium$. Finally, when θ is modified, the values of other parameters are: $\rho = 15\%$ and $\delta = 3$ hours. It is worth mentioning that changes in δ and θ are performed in a way that the aggregate duration of all local requests (ρ) remains constant. This implies that increasing δ or θ lead to fewer local requests.

The results of the experiments are studied from the practical and statistical perspectives. In statical analyses we applied T-student tests and we ensured the normality of the underlying data.

Overheads involved in dealing with VMs such as suspend/resume time, and boot up and shut down time are also considered by the scheduler and they are calculated according to Chapter 3. To measure the energy consumption of the cluster, we use the consumption information provided by the results of SPECpower benchmark[2]. Based on these information, the consumption of a resource with similar configuration is on average 117 watts, when it is utilised.

6.4.2 Experimental Results

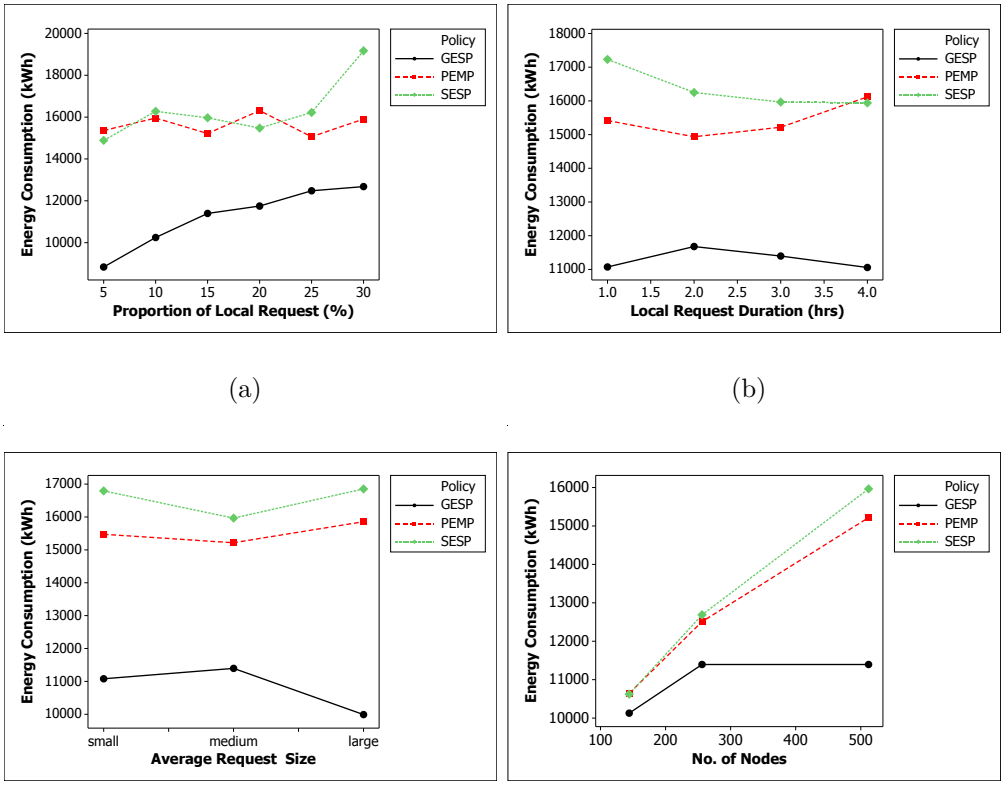
Energy Consumption

In this experiment we measure the amount of energy consumed by each policy to run the workload trace. To measure the energy consumption, we calculate the overall time that the RP's resources were switched on and we report the results in kWh.

Figure 6.3, expresses the amount of energy consumed when different policies are applied. In all subfigures of this figure, we notice that GESP leads to the lowest energy consumption since it conservatively switches on resources just when they are required by the local requests.

[2]http://www.spec.org/power-ssj2008/

More specifically, Figure 6.3(a) illustrates that the PEMP remarkably consumes less energy than SESP (around 18% or 4000 kWh) when a considerable portion of requests are local (more than 25%). However, PEMP and SESP have a similar performance when the proportion of local requests is low (less than 25%). In fact, when the proportion of local requests is low, preemption does not take place frequently, therefore, external requests have more opportunity for running. Thus, policies that try to avoid violation are not applied and result into the similar amount of consumed energy.



(a) (b)

(c) (d)

Figure 6.3: Energy consumption of different policies. The experiment is performed by modification of: (a) the percentage of time taken by local requests (ρ) where $\delta = 3$ hours and $\theta = medium$, (b) the average duration of local requests (δ) changes (in hours) where $\rho = 15\%$ and $\theta = medium$, (c) the average size of local requests (θ) where $\rho = 15\%$ and $\delta = 3$, and (d) the number of nodes in the RP varies where $\rho = 15\%$ and $\theta = medium$ and $\delta = 3$.

In Figure 6.3(b) and 6.3(c), we observe a decrease in the energy consumption of GESP. The reason is that the GESP switches on resources when there is a local request. However, when local requests are long (Figure 6.3(b)) or their size are big (Figure 6.3(c)), fewer local requests are generated, as discussed in Section 6.4.1, to keep the proportion of local requests constant. Accordingly, fewer resources are switched on and thus the energy consumption is reduced.

Additionally, in Figure 6.3(b) and 6.3(c), we notice that PEMP considerably consumes less energy than SESP. Particularly, when local requests are short (Figure 6.3(b)), the difference is more significant. T-test analysis between PEMP and SESP, in Figure 6.3(b), for durations smaller than 4 hours shows that 95% confidence interval of the average difference is (193.5, 2830.1) kWh (P-value<0.001). Also, 95% confidence interval of the average difference between PEMP and SESP, in Figure 6.3(c), is (360.8, 2030) kWh (P-value=0.04). These values suggest that the difference between PEMP and SESP is statistically and practically significant. In fact, when the local requests are small or short, more gaps remain for external requests to be scheduled. Thus, the violation risk of external requests is reduced and leads to more consolidation opportunities and less energy consumption.

In Figure 6.3(b), we notice that the energy consumption resulted from PEMP rises as the duration of the local requests increases (when the duration is 4 hours). The reason is that when the local requests are long (i.e., their average duration increases), external leases are postponed in scheduling for a long time. Therefore, resources have to remain switched on for longer time, which causes more energy consumption.

In Figure 6.3(d), we observe that GESP energy consumption changes when the number of nodes increases to 256. However, further increase of the nodes to 512 does not change its consumption. The reason is that GESP performs independently from number of nodes and the increase of consumption from 144 to 256 is because 144 nodes were not sufficient even to serve local requests. Also, we notice that the difference between PEMP and SESP becomes more significant as the number of nodes in the RP increases. This shows the efficiency of PEMP, specifically for larger RPs.

Results of the experiment, in all subfigures of Figure 6.4, reveal that PEMP is performing very close to SESP. However, the energy consumption resulted from these policies (see Figure 6.3) show that PEMP leads to less energy consumption without increasing the violation rate. Additionally, in all of the subfigures, as expected, GESP leads to very high violation rates due to switching on few resources.

In Figure 6.4(d), we observe a sharp decrease of violation rate in all policies when the number of nodes in the RP increases from 144 to 256. The reason is that 144 nodes were not sufficient to run this workload. Therefore, in all the policies we notice a high violation rate (more than 70%). However, when there are adequate number of nodes the impact of different policies is visible.

Violation Rate

This experiment measures the percentage of violations from the appointed waiting threshold. For this purpose, we report the percentage of the external requests whose waiting times were beyond the waiting threshold.

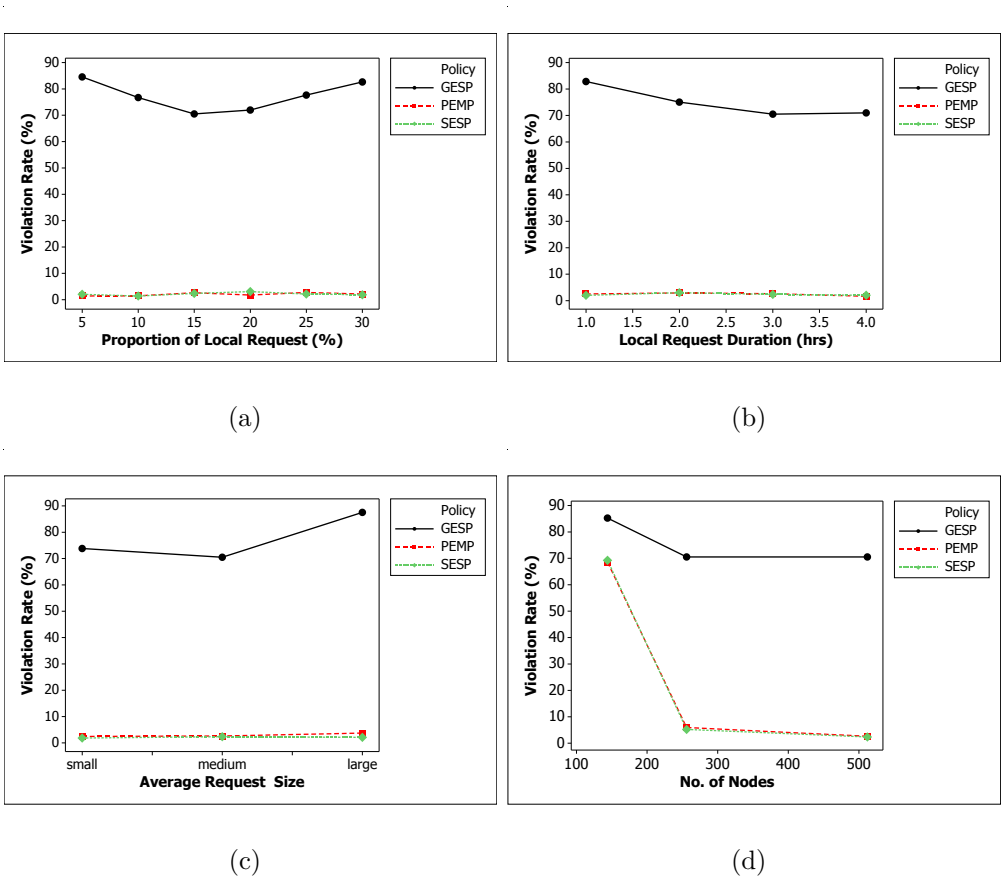


Figure 6.4: Percentage of violations from waiting threshold when different policies are applied. The experiment is performed by modification of: (a) the percentage of time taken by local requests (ρ) where $\delta = 3$ hours and $\theta = medium$, (b) the average duration of local requests (δ) changes (in hours) where $\rho = 15\%$ and $\theta = medium$, (c) the average size of local requests (θ) where $\rho = 15\%$ and $\delta = 3$, and (d) the number of nodes in the RP varies where $\rho = 15\%$ and $\theta = medium$ and $\delta = 3$.

Subfigures of Figure 6.4, express that the violation rate of SESP and PEMP are almost unchanged as ρ, δ, and θ vary. This does not mean that the violation rate is not dependent on these parameters. In fact, it explains how adaptively resources are added to the system, in a way that the violation rate does not change significantly. In other words, in these policies the number of switched on resources are changed as ρ, δ, and θ varied (see Figure 6.3) and, therefore, the violation rate does not vary significantly.

6.5 Summary

In this chapter, we investigated a contention-aware energy management policy for RPs in InterGrid. Our proposed energy management policy (PEMP) applies a fuzzy inference system to determine if the resources for a request have to be allocated through switching on resources, preemption, consolidation, or a combination of these operations.

We extended the Haizea scheduler to consider the energy consumption issues in its scheduling. Then, we implemented the proposed energy management policy (PEMP) in the Haizea and evaluated that under realistic conditions. In the next chapter, we will explain how preemption mechanism can be implemented in InterGrid to resolve the contention between local and external requests.

Chapter 7

Realising Contention-awareness in InterGrid

This chapter presents the realisation of the contention-aware scheduling in InterGrid where resource contention occurs between external and local requests within the RPs. InterGrid uses a lease-based provisioning model where each lease consists of several virtual machines. The implementation enables RPs to increase their resource utilisation through contributing resources to InterGrid without delaying local users. Additionally, several contention-aware scheduling policies are implemented and evaluated in this environment.

7.1 Introduction

InterGrid aims to provide a software platform for interconnecting islands of virtualised Grids. It provides resources in the form of VM-based leases that enable users to have customised execution environments [131].

A lease in the context of InterGrid is an agreement between resource provider and resource consumer whereby the provider agrees to allocate resources to the consumer according to the lease terms presented by the consumer [13]. InterGrid creates one lease for each user request. Virtual Machine (VM) technology is a way to implement lease-based resource provisioning [13].

RPs in InterGrid are motivated to contribute resources to the InterGrid environment to increase their resource utilisation and revenue. However, they would like to ensure that the requirements of their local requests are met. Therefore, resource provisioning in InterGrid is carried out for two types of users, namely local users and external users. Local users send their requests to the local resource management system (LRMS) of the RP to access resources. External users send

their requests to IGG to access larger amount of resources. In InterGrid, each request is contiguous and must be served with resources of a single RP.

The mixture of local and external requests within RPs of InterGrid leads to origin-initiated resource contention. The problem of origin-initiated contention in InterGrid is addressed through preemption of external requests in favour of local requests. However, VM preemption has considerable side-effects, including preemption overhead and long waiting time for external requests. However, contention-aware scheduling can proactively resolve the contention to alleviate preemption side-effects.

To realise the contention-aware scheduling in InterGrid, we consider a scenario where local and external users'leases are created on top of the physical infrastructure in form of VMs. Users can request leases with different characteristics in terms of number of VMs, memory size, and execution environment (i.e. operating system).

The work in this chapter describes the design and implementation of the contention-aware scheduling for InterGrid. The work specifically realises the architecture proposed in Chapters 1 and 3.

7.2 InterGrid Architecture

This thesis is proposed based on the InterGrid architecture, which enables resource sharing across multiple Grids. The architecture of InterGrid, which is illustrated in figure 7.1, presumes that each Grid is composed of several resource providers (RPs). RPs can be in the form of a Cluster, SMP, or a combination of them. Each RP is managed independently and has its own local users while contributes resources to InterGrid. The Local Resource Management System (LRMS) is the resource manager that handles resource provisioning for local and external requests within each RP.

Resource sharing amongst the Grids is achieved through predefined arrangements, known as peering, in addition to a coordinator for each Grid, known as InterGrid Gateway (IGG). Figure 7.1 illustrates how multiple Grids can be interconnected through InterGrid Gateways (IGGs). Peering arrangement between IGGs was initially inspired from principles of the Internet's policy-based routing. Similar idea is applied to interconnections of Grids in activities involving offload and redirection of resource requests from one Grid to another. Nonetheless, there are prominent differences between packet routing and redirection of Grid requests. While Internet routing considers only data packets, Grid inter-

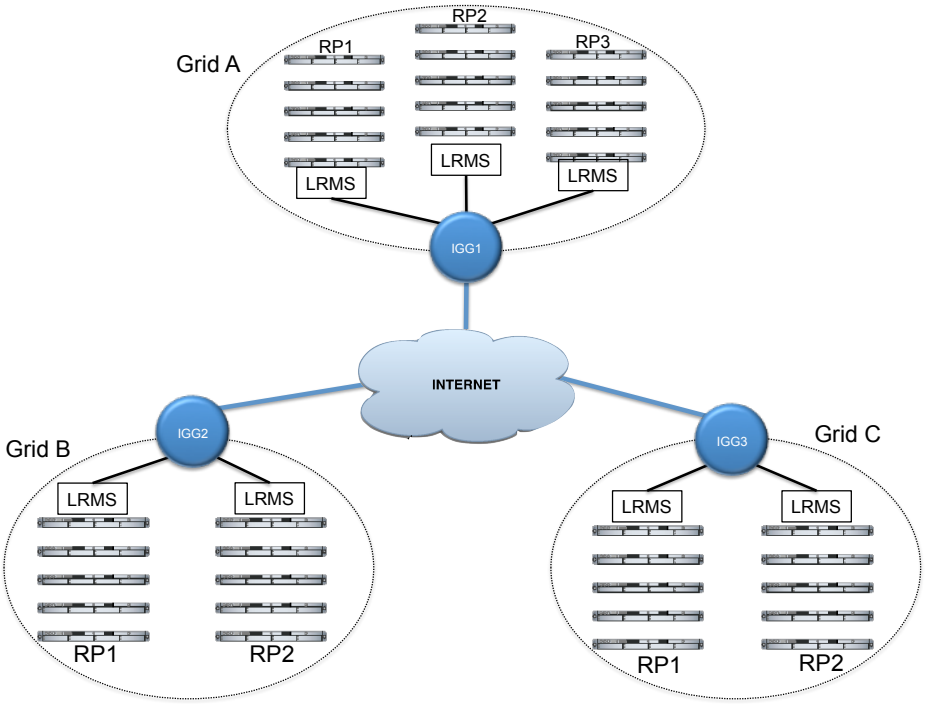


Figure 7.1: High-level view of the InterGrid components.

connection involves managing requests with numerous attributes, which implies more complexity. Furthermore, a Grid infrastructure has its policies regarding how resources are allocated to users of that Grid and to peering Grids.

Peering arrangements with other grids is handled by IGGs. This arrangement coordinates the adoption of resources from different Grids. An IGG is aware of the peering terms between Grids, selects suitable Grids that can provide the required resources, and replies to resource requests from other IGGs through allocation of resources to them. Provisioning rights over RPs inside a Grid are delegated to IGG and enable it to schedule arriving external requests on the RPs.

Distributed Virtual Environment Manager (DVE Manager) is a user level tool in the InterGrid architecture. Users who wish to access InterGrid level resources (i.e. external users) employ this tool to interact with IGG and acquire resources. The DVE Manager handles monitoring of the resources that are allocated to the user and the adaption of resource allocations based on the user application's demands.

7.2.1 IGG Structure

IGG is the core part of InterGrid and has been implemented in Java. A high-level view of its components is depicted in Figure 7.2.

One important component in this structure is the *Scheduler*, which imple-

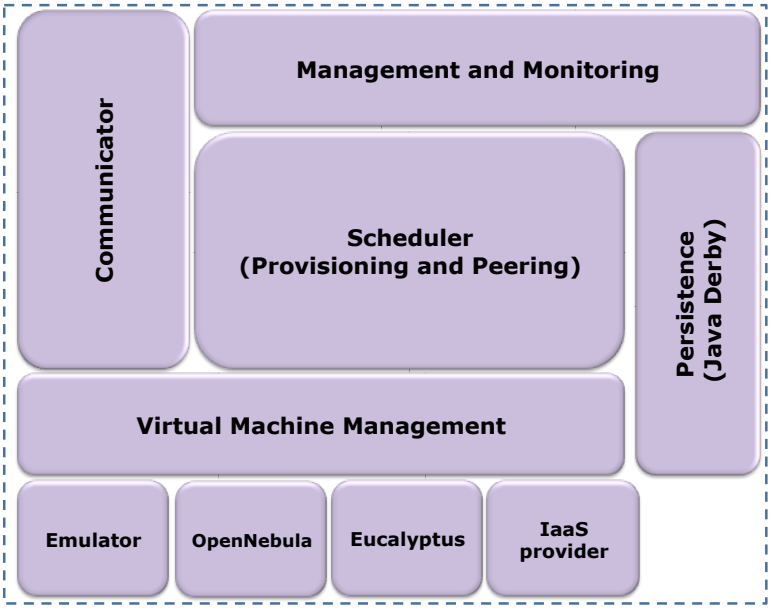


Figure 7.2: Internal structure of the InterGrid Gateway.

ments provisioning policies and peering with other IGGs. The scheduler schedules arriving external requests on the available RPs. The scheduling decisions are ordered to the RPs through the Virtual Machine Manager (VMM) interface.

VMM implementation is generic, so RPs can implement their own virtual infrastructure manager such as OpenNebula. Within the LRMS of an RP, a specific virtual infrastructure manager along with a local scheduler performs operations such as creation, start, and stop of VMs. Currently, the VMM was implemented in the LRMS to interact with OpenNebula [36] and Eucalyptus [55] to manage local resources. Moreover, interfaces have been implemented for IGG that enable it to interact with Grid'5000 [34] as a Grid middle-ware and a Cloud Amazon EC2[1] as a Cloud Infrastructure as a Service (IaaS) provider. An emulated LRMS for testing and debugging purposes was also developed.

The persistence database is used for storing information of IGG such as VM templates and peering arrangements. The Management and Monitoring provide command-line tools to configure and manage IGG. The Communication Module provides an asynchronous message-passing mechanism between IGGs, which makes IGGs loosely coupled and fault-tolerant.

7.2.2 Resource Allocation Model

In this section, we describe the resource acquisition steps for a request in InterGrid. The process is different for external and local users. The workflow for

[1]http://aws.amazon.com/ec2

resource allocation for external requests in InterGrid is illustrated in Figure 7.3, and can be described as follows:

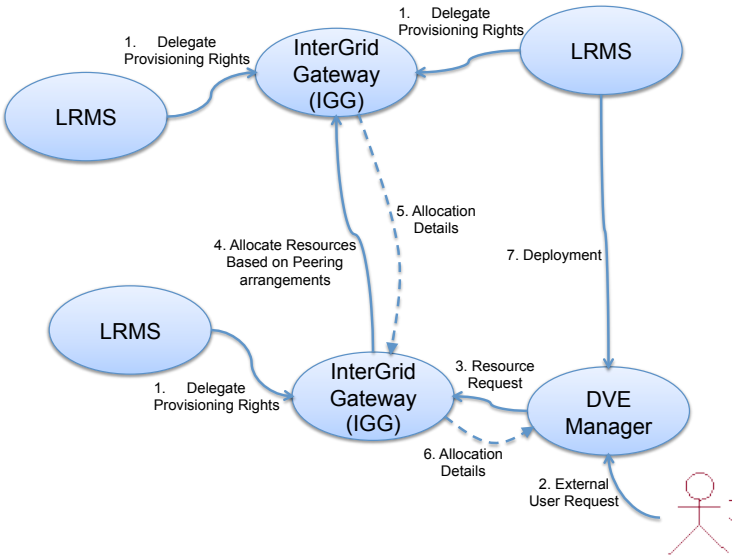


Figure 7.3: Resource allocation steps for external user requests in InterGrid.

1. Periodically, an RP advertises resources availabilities in the registry of IGG. The advertisement also implies delegation of the provisioning rights of the resource to IGG.

2. An external user initiates a request through a DVE Manager to acquire resources from InterGrid. The request is in the form of a lease that describes the required resources to deploy. Each request in InterGrid has the following characteristics:

   - Number of virtual machines.
   - Duration of the request.
   - Deadline of the request (optional).

3. If the individual Grid cannot provide the required resources possibly because of oversubscription, then IGG chooses a peering Grid, based on the peering agreements, from which the resources can be allocated.

4. Once the VMs are allocated by IGG, the DVE is given permission and other deployment information (e.g., IP address) to deploy them at the scheduled time.

The resource acquisition procedure for local users of the RP is less complicated and can be explained as follows:

1. Local users send their requests to the LRMS of their RP. This can be achieved using any user-level interface provided by each RP.

2. The local scheduler serves requests using the available resources.

3. In case the resources of the RP are leased to external requests, the LRMS preempts external leases in favour of local requests, using the policies described in Chapter 3.

7.3 System Design and Implementation

This section provides implementation details of the contention-awareness in InterGrid. We describe design and implementation choices used in InterGrid.

7.3.1 Virtual Infrastructure Manager

Since InterGrid operates based on virtualisation technology, RPs also operate based on VM resource provisioning. In our implementation, we consider the open source OpenNebula [36] as the virtual infrastructure manager to handle the VMs life-cycle across an RP. OpenNebula operates as the main component of the LRMS and provides a software layer on top of the hypervisor, and enables dynamic provisioning of resources in an RP.

The OpenNebula architecture has been designed to be flexible and modular in order to support various hypervisors and infrastructure configurations within an RP. It provides web and command-line interfaces that allows local users to conveniently request leases. For each user request, OpenNebula starts, manages, and stops VMs according to the provisioning policies in place. OpenNebula architecture includes three main elements.

The *core*, which is responsible for managing the VMs'life-cycle by performing basic operations such as start, migrate, monitoring, and terminate [164].

The *capacity manager* consists of pluggable policies that determine the VM placement across an RP. The default capacity manager in OpenNebula provides a simple VM placement and load balancing policies. In particular, it uses an *immediate* provisioning model, where virtualised resources are allocated at the time they are requested, without the possibility of requesting resources at a specific future time.

Virtualiser access drivers provide the abstraction for the underlying virtualisation layer by exposing the general functionalities of the hypervisor (e.g. start,

migrate, terminate). As a result of this component, OpenNebula is able to work with various hypervisors such as Xen, KVM, and VMware.

OpenNebula is equipped with databases for keeping VM templates. A template file consists of a set of attributes that defines a Virtual Machine. OpenNebula needs a shared storage to operate. The shared storage model requires the head node and hosts to share the VM directories and the Image Repository. Typically, these storage areas are shared using a distributed file system such as NFS [165], GlusterFS [166], and etc. A shared storage reduces VM deployment times and enables live-migration, but it can also become a bottleneck in the infrastructure and degrade the VMs performance, especially for performing disk-intensive workloads.

7.3.2 Virtualisation Infrastructure

Along with the virtual infrastructure manager, a virtualisation infrastructure (i.e. hypervisor) is required in each node of the RP to provide VMs. Specifically, the utilisation of OpenNebula enables deployment of different hypervisors in an RP.

In our implementation, we use Kernel-based Virtual Machine (KVM) [101] as the hypervisor within each node of the RPs. KVM is a hardware-assisted, fully virtualised platform for Linux on X86 hardware that has virtualisation extensions. By installing KVM, multiple execution environments (guest virtual machines from different disk images) can be created on top of each physical node. Each of these virtual machines has a private virtualised hardware, including a network card, storage, memory, and graphics adapter.

7.3.3 Scheduling in IGG

The sequence diagram of invocations in IGG classes to accept and schedule an external request is shown in Figure 7.4. For the sake of clarity, this figure just shows parts of the whole provisioning process that occurs in IGG.

When a message is received by an IGG, it is handled by a central component, called *Post Office*, which spawns one thread for each message. In the case that the received message is an external request, the spawned thread invokes the scheduler in the *Request Scheduler* class. At this stage, the scheduling is carried out by invocation of the *handleRequest* method, which extracts the request from the message and determines the appropriate RP for dispatching it. This invocation calls the appropriate method based on the configured virtual infrastructure managers (OpenNebula in our setting). In the last step, the virtual infrastructure

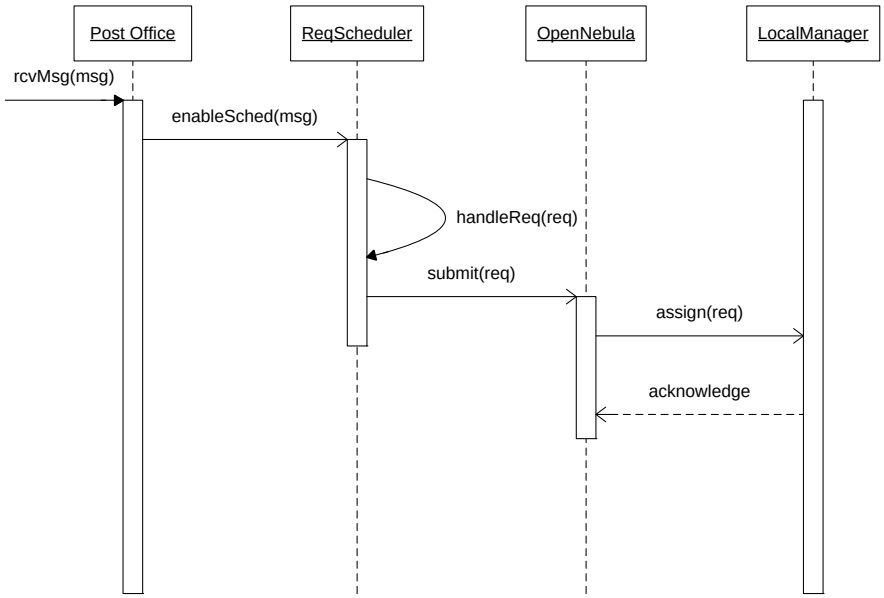


Figure 7.4: Schedule of external user requests in IGG.

manager submits the request to the local scheduler located in the RP.

7.3.4 Local Scheduler

The critical part of an LRMS is a scheduler that has to allocate resources across an RP efficiently. The local scheduler should be aware of the contention between local and external requests within an RP.

As mentioned earlier, OpenNebula, as the virtual infrastructure manager in the RPs, offers an immediate provisioning model, where virtualised resources are allocated at the time they are requested. However, resource provisioning in InterGrid implies requirements that cannot be supported within this model, such as resource requests that are subject to priorities, capacity reservations at specific times, and variable resource usage throughout a VM's lifetime. Additionally, in smaller RPs not all requests can be allocated immediately due to resource shortage.

Haizea is an open source scheduler developed by Sotomayor et al. [17] that employs VM-based leases for resource provisioning. The advantage of Haizea is that is considers overheads of deploying VMs (e.g., suspension and resumption) in the scheduling. It enables resource providers to provide advance-reservation leases (to guarantee resource availability) along with best-effort leases (to increase resource utilisation) where advance-reservation leases have preemptive priority over best-effort leases.

We adopt Haizea as the local scheduler of the LRMS in RPs. As a result, the

scheduling capability of the virtual infrastructure manager (i.e. OpenNebula) is extended and enables the LRMS to recognise the contention between local and external requests that occurs in the RP. More importantly, adopting Haizea as the local scheduler enables lease of resources to external requests in a best-effort manner while respecting allocation of resources to local requests in their requested time interval. When a contention occurs, the scheduler resolves it through preemption of external lease(s) and vacation of resources to serve the local request. In this way, the local scheduler operates as the scheduling back-end of OpenNebula. It also employs backfilling scheduling strategy along with VMs'management abilities (i.e., suspend, resume, and migrate) to efficiently schedule the leases and increase the resource utilisation.

Although the local scheduler described enables recognition of the contention between local and external requests and resolves it using preemption, the policy used does not consider the side-effects caused by preemption. Therefore, in the next step, we implemented different preemption policies (as discussed in Chapter 3) in the local scheduler that proactively detects the resource contentions and try to reduce their impact. These policies decrease the number of resource contentions take place and increase the resource utilisation in an RP. We have implemented the following contention-aware preemption policies for the local scheduler:

- MLIP (Naïve): This policy tries to minimise the contention by reducing the number of requests affected by the preemption. Thus, this policy preempts large leases regardless of the overhead imposed for their preemption.

- MOV: The second preemption policy that we have implemented sought to minimise the overall overhead time imposed to the system by preempting VMs. Implementation of this policy is based on the selection of a set of leases for preemption that result in the minimum overhead time. For such purpose, we calculate the overhead imposed by preemption of each lease, then preempt leases with minimum overhead.

- MOML: This policy takes into account both the number of contentions as well as the overhead time imposed to the system by preemption of different leases. Implementation of this policy involves two rounds. In the first round, the overhead imposed by preempting each external lease is calculated. In the second round, leases are sorted based on the imposed overhead, then, the minimum number of leases are selected by considering the overhead of preempting each lease.

The sequence diagram of invocations between local scheduler classes is shown in Figure 7.5. The scheduling process in the local scheduler starts by receiving a

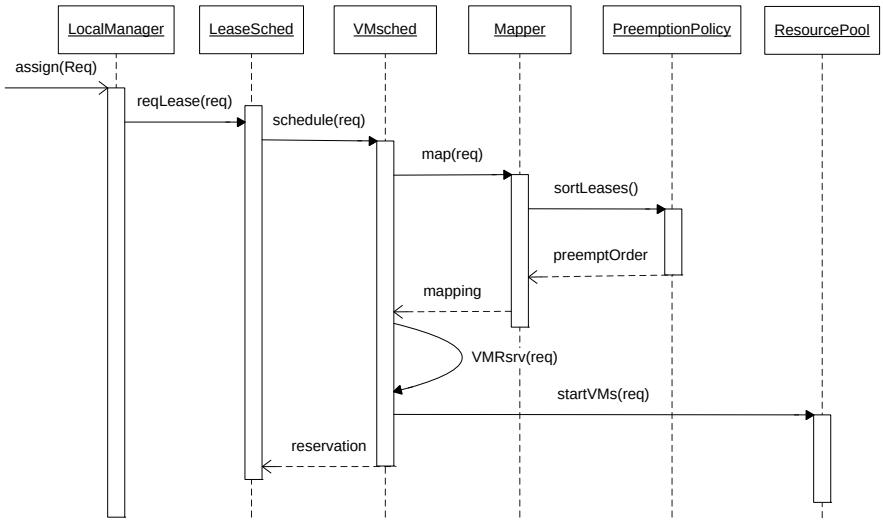


Figure 7.5: Schedule of local requests in the local scheduler.

lease request either from local or external user (through IGG) in the *LocalManager* class.

The manager requests the *LeaseScheduler* class to schedule the lease request. Then, the *schedule* method in the *VMScheduler* class is called which schedules local and external requests. For local requests VMs are scheduled based on the requested time interval. External requests are allocated in the first vacant space. The *map* function in the *mapper* class maps requested resources to the physical resources based on their availability times. When the mapper class handles a local requests, if there is not enough resources, then the mapper calls the *PreemptionPolicy* to determine the preferred order of preempting external leases. The order is determined based on the preemption policy discussed above. Then, the mapper can perform the mapping and returns the mapping list to the *VMScheduler*. Using the mapping information, the VMScheduler calls the *VMRsrv* and updates the scheduling information of the resources. After that, the lease can be started by calling the *startVMs* method in the *ResourcePool* class. Additionally, the *LeaseScheduler* is informed to update all the affected leases in the scheduling table.

7.4 Performance Evaluation

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

- A four-node cluster as the RP. Worker nodes are 3 IBM System X3200 M3 machines, each with a quad-core Intel Xeon x3400, 2.7 GHz processor and 4 GB memory. The head node, where the LRMS resides, is a Dell Optiplex

755 machine with Intel Core 2 Duo E4500, 2.2 GHz processor and 2 GB of memory.

- The host operating system installed in the server nodes is the CentOS 6.2 Linux distribution. Also, the operating system in the head node is Ubuntu 12.4.
- All the nodes are connected through a 100 Mbps switched Ethernet network.
- We used OpenNebula 3.4 and Haizea version 1.1 as the virtual infrastructure manager and the local scheduler, respectively.
- Qemu-KVM 0.12.1.2 is used as the hypervisor on each server.
- GlusterFS is used as the cluster file system. It aggregates commodity storages across a cluster and forms a large parallel network file system [166]. The disk images needed by the VMs and the memory image files (created when a VM is suspended) are stored on the shared file system.

The scenario we consider in our experiment involves an InterGrid with 3 IGGs with peering arrangements established between them, as illustrated in Figure 7.6. IGG1 has the cluster as the RP and users from IGG2 and IGG3 request leases through their DVE manager. Based on the peering arrangements, IGG1 provides them resources. IGG1 receives these requests in form of external requests and they are allocated resources through the LRMS of the RP. However, the RP has its own local requests that have more priority than the external ones. Information of the lease requests received by the LRMS are explained in the Table 7.1.

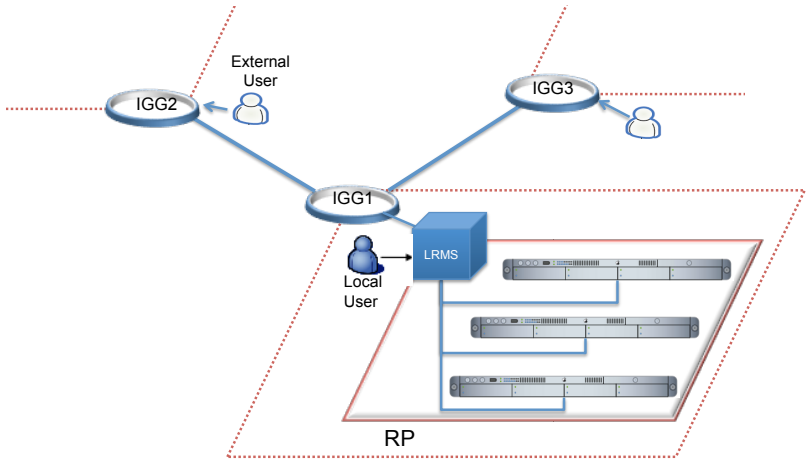


Figure 7.6: Evaluation scenario based on 3 InterGrid Gateways.

To be able to follow the order of events occurring in the system and demonstrate their impact, we perform the evaluation on 7 lease requests that are submit-

ted to the RP. Each row of the table shows the arrival time, number of requested processing elements, amount of memory, duration, and request type (i.e., local or external). We consider 00:00:00 as the start of the experiment (i.e., the arrival of the first request) and the arrival time of other requests are proportional to the start time of the experiment. All of these lease requests use a ttylinux disk image located on the shared storage.

Table 7.1: Characteristics of lease requests used in the experiments.

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

7.4.1 Evaluation Results

In the first experiment, we demonstrate how our implementation enables InterGrid to resolve the origin-initiated contention between local and external requests. It shows the effect of preempting existing external leases on a virtualised physical testbed to satisfy the requirements of an arriving local request. For such purpose, we compare the situation where there is not any preemption policy (NOP) against the situation where the MOML preemption policy is applied. In the former, the local request (request ID: 7) is rejected, whereas in the latter, external leases (request ID: 5, 6) are preempted and vacate resources for the local requests. We notice that the local request request is served without being delayed. Additionally, the RP could utilise its resources more efficiently by leasing them to the external requests.

More specifically, Figure 7.7 indicates how the MOML contention resolution policy allocates resources to the local request in comparison with NOP. In fact, the vertical axis in this figure shows how the resource utilisation varies while the workload is running while the horizontal axis presents the overall makespan of the workload execution.

In the beginning, the resource utilisation rapidly increases to 100% for both policies due to allocation of resources to the arriving external requests (requests 1 to 6 in Table 7.1). As the time passes, we observe that the resource utilisation gradually drops to 0% as the requests are completed. However, we can see that the resource utilisation reduction is sharper for NOP than MOML. Indeed, when the resources are 100% utilised and the local request arrives, the MOML policy

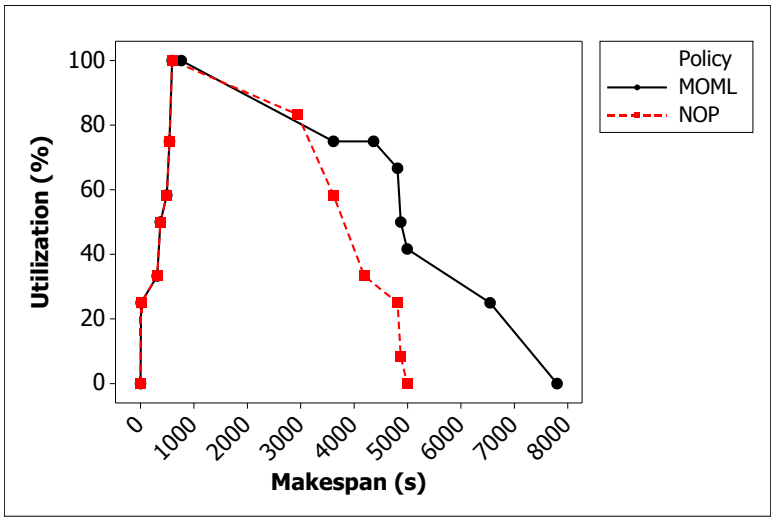


Figure 7.7: Illustration of contention resolution in InterGrid using MOML policy against a scenario where no contention resolution is applied (NOP). The vertical axis shows how the resource utilisation varies over the time using these policies. The horizontal axis shows the overall makespan to execute the requests.

preempts external requests and schedules them after the local request. After completing the local request, the preempted external requests are resumed. As a result, the MOML policy operates with high utilisation for longer time to run the local request. Additionally, as it is presented in the Figure 7.7, preemption of external request in favour of local requests, and resuming at a later time leads to longer makespan for the MOML policy. We expect that an increase in the number of local requests raises the resource utilisation as well as makespan of the external requests.

As discussed in Chapter 3, various contention resolution policies (preemption policies) preempt different leases that result in different amount of resource contention and overhead time. Hence, in the second experiment we evaluate the efficacy of the implemented policies from the overhead and resource contention aspects. Specifically, we measure how many resource contentions are resulted from different policies and how the makespan is affected in each policy. Additionally, we determine the overhead imposed to the system by suspension and resumption operations on the preempted leases.

In Table 7.2 the number of resource contentions as well as the amount of overhead resulted from MLIP, MOV, and MOML policies are listed. As we can see, the MLIP policy affects 2 leases and the overall size of memory should be written/read to/from memory is 1152 MB. Knowing that the read/write throughput of our Gluster file system is 40MB/s [166], the overhead of suspending and

Table 7.2: Number of resource contention (lease preemption), overhead, and makespan resulted from applying different preemption policies.

Policy	**Preempted Leases**	**Overhead (s)**	**Makespan (s)**
MLIP	$\{1, 6\}$	57.6	7800
MOV	$\{5, 2, 3\}$	25.6	8479
MOML	$\{5, 6\}$	25.6	7800

resuming these leases is 57.6 seconds. The MOV policy aims at minimising the overall preemption overhead. Therefore, it preempts leases that impose minimum overhead to the system (i.e., $\{5, 2, 3\}$) and the amount of memory that is de-allocated and snapshot on the disk is 512 MB, which implies 25.6 seconds overhead. MOML affects just two leases ($\{5, 6\}$) while results in 512 MB of memory suspension and resumption overhead. The comparison of the results from different policies indicates that the selection of different preemption policies affects the number of contentions and time overhead imposed to the system.

7.5 Summary

This chapter presented the realisation of the contention resolution in InterGrid where local and external leases coexist in RPs. The system prototype shows how an RP can increase its resource utilisation by accepting external requests without affecting the local requests. The provided implementation also realises the contention-aware preemption policies for the RPs discussed in Chapter 3. Evaluation of these preemption policies indicated the impact of these policies on the number of resource contentions as well as amount of imposed overhead.

Chapter 8

Conclusions and Future Directions

This chapter summarises the research work as well as the major findings of this dissertation. Moreover, research topics that have emerged during this research but have not been addressed in this dissertation are discussed.

8.1 Discussion

At the outset of this dissertation, we focused on a general challenge: requests from different origins contend to access resources in a federated Grid environment. Particularly, the case of InterGrid, where resources are provided based on virtual machines (VMs), was considered. In InterGrid, resource contention occurs between local and external requests within each resource provider and local requests have priority over external ones. We approached the challenge by considering preemption mechanism to resolve the contention and to meet requirements of local (i.e., high priority) requests. However, preemption mechanism affects the performance of external (i.e., low priority) requests. Therefore, side-effects of preemption mechanism, including long waiting time for external requests, as well as imposed overhead of preemption were taken into consideration in the proposed solutions.

In this regard, this dissertation explored research works undertaken on contention management in distributed systems, including Clusters, Grids, and Clouds, and enumerated several characteristics of existing mechanisms, such as their operational model, architectural views, context, type of resource contention, and placement of the mechanism in the resource management system. This exploration revealed:

- Various types of resource contentions and approaches to resolve them.
- The impact of different elements of a resource management system in resolving the contention.
- The applicability and challenges of preemption mechanism to resolve different types of resource contentions.

The exploration also revealed a lack of a comprehensive scheme that considers all side-effects of resource contention and meets the requirements described earlier.

Based on the lessons learned, we recognised the potential impact of local scheduling, global scheduling (meta-scheduling), admission control, and energy management units on the resource contentions in a resource management system. We also learned that emergence of resource provisioning based on virtual machine technology has posed the preemption as a predominant solution for resource contentions.

Therefore, we proposed a comprehensive contention management scheme that handles the resource contention through different components of resource management system. This scheme includes two main strategies; the first strategy avoids contentious situation by establishing contention-awareness in the scheduling of users' requests. The second strategy handles side-effects of resource contention mainly in terms of long waiting time for external requests. Specifically, the second strategy considers two main circumstances that lead to long waiting time for external requests. One situation deals with the scenario where there is a surge in demand from local users in resource providers. The other one deals with situation that energy management mechanisms are applied within the resource providers. The proposed strategies reside in the local scheduler, meta-scheduler (InterGrid Gateway), admission control, and energy management units of the resource management system.

We first presented the feasibility of the VM preemption approach in resolving resource contention between local and external requests within the local scheduler of resource providers. We recognised two side-effects caused by the preemption mechanism, that are increase in the waiting time of external requests and overhead time imposed to the system. To understand the overhead time of preempting VMs, we modelled the overhead time imposed for performing different operations on the preempted VMs, such as suspension, and migration.

We also noticed that preemption of different external requests impacts the number of request contentions takes place, waiting time of external requests, and overall overhead time imposed to the system. Considering these impacts, we proposed preemption policies that determine appropriate requests for preemption.

Specifically, we proposed the MOML policy that mitigates both the imposed overhead time and the number of resource contentions.

To avoid resource contention, in the gateway level (IGG), we proposed a probabilistic scheduling policy, called workload allocation policy that proactively distributes external requests on different resource providers in a way that reduces the number of resource contentions occurring in a Grid. Moreover, we investigated a situation where some of external requests are more valuable than others and we would like to further decrease the likelihood of preemption for them. Thus, a dispatch policy is proposed along with the scheduler that regulates the dispatching order of external requests in a way that the probability of preemption for more valuable external requests is decreased.

Results of the comparison with other scheduling policies indicate that the workload allocation policy, specifically when it is applied along with the dispatch policy, significantly (at least 60%) decreases the number of resource contentions. Additionally, the workload allocation policy along with the dispatch policy significantly reduces the likelihood of preemption for more valuable external requests.

Preemption mechanism resolves the resource contention. However, it increases the risk of long waiting time for external (low priority) requests. This particularly can occur when there is a surge in demand from external and local users. To manage this side-effect, we proposed a policy in the admission control unit of resource providers that determines the ideal number of external requests that can be accepted and completed within a limited waiting time and without being starved. Experimental results indicated that the proposed policy significantly reduces the rate of long waiting time for external requests (up to 20%) comparing to a situation where all requests are accepted. Additionally, this policy leads to completion of more external requests (up to 25%) comparing with a situation where external requests are accepted conservatively.

In contrast to circumstances that there is a surge in demand, when the resource providers operate at low utilisation they desire to decrease their energy consumption. However, decreasing the number of active resources increases resource contention and its side-effects, mainly in terms of long waiting time for external requests. To counter such effect, we proposed a contention-aware energy management policy that adapts the energy consumption of a resource provider based on users' performance demands and without causing long waiting time for external requests. For an arriving request, the policy determines if the resources should be allocated through switching on resources, preemption of current external requests, consolidation, or a combination of these operations. Experimental results reveal that the policy reduces the energy consumption in a resource

provider up to 18% (4000 kWh), over the course of 30 days, and without significant violation from waiting threshold for external requests, when compared to other baseline policies.

We realised the preemption-based contention management scheme by presenting a system prototype in InterGrid. The system demonstrates how a resource provider can increase the utilisation of its resources by accepting external requests and preempting them when there are not sufficient resources to serve local requests. The prototype also realises the contention-aware preemption policies within the resource provider. Evaluation of these preemption policies indicated the impact of these policies on number of resource contentions as well as amount of imposed overhead.

8.2 Future Directions

The focus of this dissertation was on contention management in virtualised federated Grids. There are still open issues that have not been addressed in this dissertation and can serve as a starting point for future research.

8.2.1 Contention-aware Peering Policy

This dissertation studied contention-aware scheduling in the local scheduler level and meta-scheduling (i.e., IGG) levels. It would be relevant to consider contention-aware scheduling between Grids.

As mentioned earlier, in InterGrid resource sharing between Grids is achieved based on pre-defined peering agreements that denote the conditions for resource sharing. In a practical setting, an InterGrid Gateway (IGG) has peering arrangements with several other IGGs. In this situation, an IGG should choose the best peer Grid to redirect a given request. Currently, IGG chooses the provider that offers earlier start time for the request. However, it does not consider the probability of resource contention in the destination Grid. A contention-aware peering policy that proactively schedules requests on other Grids by considering the likelihood of resource contention in the destination Grid can be effective and is worth of investigation.

8.2.2 Contention Management for Workflow Applications

This dissertation proposes contention resolution mechanisms for situation where requests are independent of each other.

One important case to investigate is a system where users' requests are in form of workflow applications [167], where a precedence order exists between tasks. Therefore, a task cannot start until all its parent tasks complete. Running a workflow application requires creation of multiple requests, where there are dependencies between them. Contention management in such a system requires investigation of specific policies that consider the dependency between multiple requests.

8.2.3 Price-based Contention Management Policies

Although we studied the resource contention challenge in the InterGrid context, the problem exists in any form of distributed system that supports distinct QoS levels for users' requests. Particularly, the resource contention problem exists in Cloud computing (IaaS[1] providers) where there are certain priorities and pricing between their different service levels. For instance, Amazon EC2[2] provides Spot, On-demand, and Reserved VM instances. Spot instances in Amazon EC2 can be terminated (canceled), if the price goes beyond the user bid. In a smaller Cloud (e.g., a private Cloud), Spot VM instances can potentially be terminated in favour of On-demand or Reserved requests [124]. In these circumstances, application of an appropriate contention-aware scheduling policy can help in reducing the number of VM preemptions, which results in more user satisfaction and increases resource utilisation and revenue for the Cloud provider. Therefore, contention-aware scheduling policies are required to optimally fulfill this demand of Cloud providers.

Another possibility towards priced-based policies can be exploring admission control mechanisms that decide whether or not a low priority request can be accepted or not. An example of a similar work in the area is the research undertaken by Percival et al. [42], where a resource intensive request is accepted if it can compensate the loss of earning resulting from not admitting several small requests.

8.2.4 Contention Management for Co-allocated and Adaptive Requests

This dissertation considers situation that the requests are rigid and have to be served within one resource provider. Accordingly, for preemption, the whole request has to be preempted.

[1]Infrastructure as a Service
[2]http://aws.amazon.com/ec2

A possible future direction is relaxing these assumptions. Relaxing the first assumption implies having Moldable [168] and Malleable [169] requests. In the former, the number of required resources can be determined at its start time. In the latter, the number of allocated resources can vary during its execution. Resolving resource contention in both cases requires new mechanisms due to the features of these requests. Particularly, malleable requests enable performing partial preemption of requests, which affects the way contention management policies operate.

By relaxing the second assumption, requests can be co-allocated [170], which means serving requests with resources from several providers at the same time. Contention management and preemption for these requests needs coordination of different parts of a request co-allocated on several resource providers. Additionally, the cost of performing various preemption scenarios, such as suspension and migration, should be determined for this circumstance.

8.2.5 Grid Level Admission Control

In Chapter 5, an admission control mechanism was applied in the resource provider level. The idea of applying admission control mechanisms can be extended to the InterGrid Gateway (IGG) level. In this circumstance, the admission control mechanism would be able to prevent resource contentions by not accepting requests from other peer Grids. In fact, such admission control mechanism adapts the amount of resources that can be offered to peer Grids to the workload condition of the resource providers.

Another usage of such mechanism can be resolving the inter-domain-initiated contention (as describe in Chapter 2) between peer Grids.

8.2.6 Dynamic Preemption Decisions

In Chapter 3, possible scenarios for VM preemption, such as suspension and migration, were discussed. In fact, in that case the operation to be performed on the preempted VM was determined based on the type of user request.

As a future direction, it would be interesting to investigate mechanisms that dynamically decide about the appropriate operation to be performed on a preempted VM. This decision can be based on characteristics of the request and the system. For instance, for a data-intensive request, it might be better to suspend and queue it in the source provider rather than migrate it to another provider. Nonetheless, for a request whose QoS demands cannot be met in the

source provider, it would be useful to migrate it to a more powerful provider to satisfy its QoS demand.

References

[1] M. De Assunção, R. Buyya, and S. Venugopal, "InterGrid: A case for internetworking islands of Grids," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 8, pp. 997–1024, 2008.

[2] M. de Assunçao, "Provisioning techniques and policies for resource sharing between grids," Ph.D. dissertation, University of Melbourne, Department of Computer Science and Software Engineering, 2009.

[3] 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)*, 2008, pp. 157–168.

[4] B. Lawson and E. Smirni, "Multiple-queue backfilling scheduling with priorities and reservations for parallel systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 4, pp. 40–47, 2002.

[5] K. Li, "Optimal load distribution in nondedicated heterogeneous cluster and grid computing environments," *Journal of System Architecture*, vol. 54, pp. 111–123, 2008.

[6] R. Novaes, P. Roisenberg, R. Scheer, C. Northfleet, J. Jornada, and W. Cirne, "Non-dedicated distributed environment: A solution for safe and continuous exploitation of idle cycles," *Scalable Computing: Practice and Experience*, vol. 6, no. 3, pp. 15–26, 2001.

[7] L. Gong, X. Sun, and E. Watson, "Performance modeling and prediction of nondedicated network computing," *IEEE Transactions on Computers*, vol. 51, no. 9, pp. 1041 – 1055, 2002.

[8] 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*, vol. 5, no. 2, pp. 173–195, 2007.

[9] P. Ruth, P. McGachey, and D. Xu, "Viocluster: Virtualization for dynamic computational domain," in *Proceedings of International IEEE Conference on Cluster Computing (Cluster'05)*, USA, 2005, pp. 1–10.

[10] 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)*, 2002, pp. 24–37.

[11] J. Moore, D. Irwin, L. Grit, S. Sprenkle, and J. Chase, "Managing mixed-use clusters with cluster-on-demand," Duke University Department of Computer Science, Tech. Rep., 2002.

[12] 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)*, USA, 2010, pp. 658–666.

[13] 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*, USA, 2008, pp. 87–96.

[14] M. Zhao and R. Figueiredo, "Experimental study of virtual machine migration in support of reservation of cluster resources," in *Proceedings of the 3rd International Workshop on Virtualization Technology in Distributed Computing*, 2007, pp. 5–11.

[15] 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)*, 2009, pp. 1–10.

[16] M. Amini 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)*, vol. 72, no. 2, pp. 231–245, 2012.

[17] 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*, USA, 2009, pp. 59–68.

[18] A. Iosup, "A framework for the study of grid inter-operation mechanisms," Ph.D. dissertation, Delft University of Technology, 2009.

[19] L. Field and M. Schulz, "Grid interoperability: the interoperations cookbook," in *Journal of Physics: Conference Series*, vol. 119. IOP Publishing, 2008, pp. 120–139.

[20] 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*, vol. 3, pp. 429–439, 2007.

[21] 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*, vol. 18, no. 6, pp. 685–699, 2006.

[22] S. Gabriel, "Gridka tier1 site management," *International Conference on Computing in High Energy and Nuclear Physics (CHEP'07)*, pp. 94–103, 2007.

[23] 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'07)*, 2007.

[24] P. Shih, H. Chen, Y. Chung, C. Wang, R. Chang, C. Hsu, K. Huang, and C. Yang, "Middleware of Taiwan UniGrid," in *Proceedings of the ACM Symposium on Applied Computing*. ACM, 2008, pp. 489–493.

[25] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of High Performance Computing Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[26] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: The condor experience," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[27] F. Sacerdoti, M. Katz, M. Massie, and D. Culler, "Wide area cluster monitoring with ganglia," in *Proceedings of International Conference on Cluster Computing*, 2003, pp. 289–298.

[28] 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 (HPDC'01)*, 2001, pp. 181–194.

[29] T. Hey and A. E. Trefethen, "The UK e-science core programme and the Grid," *Future Generation Computer Systems (FGCS)*, vol. 18, no. 8, pp. 1017 – 1031, 2002.

[30] R. Kubert and S. Wesner, "Service level agreements for job control in high-performance computing," in *International Multiconference on Computer Science and Information Technology*, 2010, pp. 655 –661.

[31] R. Schneider, G. Faust, U. Hindenlang, and P. Helwig, "Inhomogeneous, orthotropic material model for the cortical structure of long bones modelled on the basis of clinical ct or density data," *Computer Methods in Applied Mechanics and Engineering*, vol. 198, no. 27-29, pp. 2167 – 2174, 2009.

[32] 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*, vol. 26, no. 7, pp. 979–985, 2010.

[33] 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)*, 2004, pp. 176–193.

[34] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab *et al.*, "Grid'5000: a large scale

and highly reconfigurable experimental Grid testbed," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, p. 481, 2006.

[35] D. Jackson, Q. Snell, and M. Clement, "Core Algorithms of the Maui Scheduler," in *Job Scheduling Strategies for Parallel Processing*, 2001, vol. 2221, pp. 87–102.

[36] 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*, USA, 2008.

[37] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[38] J. Walters, B. Bantwal, and V. Chaudhary, "Enabling interactive jobs in virtualized data centers," *Cloud Computing and Applications*, vol. 1, 2008.

[39] 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*, vol. 71, no. 6, pp. 732 – 749, 2011.

[40] A. Barak and A. Shiloh, "The mosix2 management system for linux clusters and multi-cluster organizational grids," Hebrew University of Jerusalem, Tech. Rep., 2007.

[41] P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh, "Spruce: A system for supporting urgent high-performance computing," *Grid-Based Problem Solving Environments*, pp. 295–311, 2007.

[42] 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*, vol. 21, pp. 133–158, 2009.

[43] 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*, 2006, pp. 291–302.

[44] 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)*, 2008, pp. 187–194.

[45] 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*, Brazil, 2007, pp. 83–91.

[46] M. 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*, 2010, pp. 351–362.

[47] 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)*, 2009, pp. 141–150.

[48] 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)*, 2005, pp. 13–20.

[49] A. Oram, Ed., *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* Sebastopol, CA, USA: O'Reilly & Associates Inc., 2001.

[50] Y. Kwok, 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, 2005, pp. 1143–1150.

[51] S. Ontañón and E. Plaza, "Cooperative case bartering for case-based reasoning agents," *Topics in Artificial Intelligence*, pp. 294–308, 2002.

[52] T. Sandholm, K. Lai, and S. Clearwater, "Admission control in a computational market," in *Proceedings of 8th IEEE International Symposium on Cluster Computing and the Grid*, 2008, pp. 277–286.

[53] M. Amini 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)*, Australia, 2011, pp. 25–34.

[54] 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)*, 2011, pp. 419–432.

[55] 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)*, 2009, pp. 124–131.

[56] 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*, 2003, pp. 1–20.

[57] 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)*, 2007, pp. 1–12.

[58] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.

[59] E. Walker, J. Gardner, V. Litvin, and E. 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)*, 2006, pp. 95–103.

[60] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron, "Virtual workspaces in the grid," *Proceeding of the 11th International Euro-Par Conference on Parallel Processing*, pp. 421–431, 2005.

[61] M. D. Assunção and R. Buyya, "Architectural elements of resource sharing networks," *Handbook of Research on Scalable Computing Technologies*, pp. 517–550, 2009.

[62] A. Barak, A. Shiloh, and L. Amar, "An organizational grid of federated mosix clusters," in *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 05)*, vol. 1, 2005, pp. 350–357.

[63] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science clouds: Early experiences in cloud computing for scientific applications," *Proceeding of Cloud Computing and Applications*, vol. 1, 2008.

[64] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao *et al.*, "From virtualized resources to virtual computing grids: the in-vigo system," *Future Generation Computer Systems*, vol. 21, no. 6, pp. 896–909, 2005.

[65] I. Krsul, A. Ganguly, J. Zhang, J. Fortes, and R. Figueiredo, "Vmplants: Providing and managing virtual machine execution environments for grid computing," in *Proceedings of the ACM/IEEE Conference Supercomputing*, 2004, pp. 7–17.

[66] 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*, 2006, pp. 440–451.

[67] 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*, 2008, pp. 243–264.

[68] 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.

[69] R. Nathuji and K. Schwan, "Virtualpower: coordinated power management in virtualized enterprise systems," in *ACM SIGOPS Operating Systems Review*, vol. 41(6), 2007, pp. 265–278.

[70] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010, pp. 39–50.

[71] A. Verma, P. Ahuja, and A. Neogi, "Power-aware dynamic placement of HPC applications," in *Proceedings of the 22nd Annual International Conference on Supercomputing*, 2008, pp. 175–184.

[72] L. Grit, L. Ramakrishnan, and J. Chase, "On the duality of jobs and leases," Duke University, Department of Computer Science, Tech. Rep. CS-2007-00, April 2007.

[73] M. Xu, "Effective Metacomputing using LSF MultiCluster," in *Proceedings of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2001, pp. 100–105.

[74] B. Nitzberg, J. M. Schopf, and J. P. Jones, "Grid resource management." Kluwer Academic Publishers, 2004, ch. PBS Pro: Grid computing and scheduling attributes, pp. 183–190.

[75] 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*, 2008, pp. 116–131.

[76] W. Smith, I. Foster, and V. Taylor, "Scheduling with advanced reservations," in *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, 2000, pp. 127–132.

[77] M. Zwahlen, "Managing contention in collaborative resource sharing systems using token-exchange mechanism," Ph.D. dissertation, Royal Institute of Technology (KTH), Stockholm, Sweden, 2007.

[78] 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*, 2008, pp. 97–108.

[79] 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)*, 2008, pp. 13–20.

[80] 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)*, 2008, pp. 41–57.

[81] A. Sodan, "Service control with the preemptive parallel job scheduler scojopect," *Journal of Cluster Computing*, vol. 14, pp. 165–182, 2011.

[82] 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*, 2009, pp. 261–276.

[83] D. Tsafrir, Y. Etsion, and D. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, 2007.

[84] 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)*, vol. 3, no. 2/3, pp. 122–152, 2005.

[85] "Bright cluster manager." http://www.brightcomputing.com.

[86] H. Shachnai, T. Tamir, and G. Woeginger, "Minimizing makespan and preemption costs on a system of uniform machines," *Algorithmica Journal*, vol. 42, no. 3, pp. 309–334, 2005.

[87] E. Parsons and K. Sevcik, "Implementing multiprocessor scheduling disciplines," in *Proceedings of 3rd International Workshop on Job Scheduling Strategies for Parallel Processing*, 1997, pp. 166–192.

[88] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '11)*, USA, 2011, pp. 121–132.

[89] Y. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Transaction Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506–521, 1996.

[90] A. Ferrari, S. Chapin, and A. Grimshaw, "Heterogeneous process state capture and recovery through process introspection," *Cluster Computing*, vol. 3, no. 2, pp. 63–73, 2000.

[91] K. Chanchio and X. Sun, "Data collection and restoration for heterogeneous process migration," *Software: Practice and Experience*, vol. 32, no. 9, pp. 845–871, 2002.

[92] J. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent checkpointing under unix," in *Proceedings of the USENIX Technical Conference*. USENIX Association, 1995, pp. 18–28.

[93] M. Litzkow and M. Solomon, "The evolution of condor checkpointing," in *Mobility: Processes, Computers, and Agents*. ACM Press/Addison-Wesley Publishing Co., 1999, pp. 163–174.

[94] P. Hargrove and J. Duell, "Berkeley lab checkpoint/restart (BLCR) for linux clusters," in *Journal of Physics: Conference Series*, vol. 46. IOP Publishing, 2006, pp. 494–502.

[95] H. Zhong and J. Nieh, "Crak: Linux checkpoint/restart as a kernel module," Department of Computer Science, Columbia University, Tech. Rep., 2001.

[96] J. Heo, S. Yi, Y. Cho, J. Hong, and S. Shin, "Space-efficient page-level incremental checkpointing," in *Proceedings of the ACM symposium on Applied Computing*, 2005, pp. 1558–1562.

[97] S. Jones, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Antfarm: Tracking processes in a virtual machine environment," in *Proceedings of the USENIX Annual Technical Conference*, 2006, pp. 1–14.

[98] K. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 1, pp. 63–75, 1985.

[99] G. Stellner, "CoCheck: Checkpointing and process migration for MPI," in *Proceedings of the the 10th International Parallel Processing Symposium*, 1996, pp. 526–531.

[100] 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*, 2006, pp. 127–1233.

[101] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: the linux virtual machine monitor," in *Linux Symposium*, 2007, pp. 225–232.

[102] 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*, vol. 37, no. 5, pp. 164–177, 2003.

[103] K. Chanchio, C. Leangsuksun, H. Ong, V. Ratanasamoot, and A. Shafi, "An efficient virtual machine checkpointing mechanism for hypervisor-based HPC systems," in *Proceedings of the High Availability and Performance Computing Workshop (HAPCW)*, 2008, pp. 29–35.

[104] W. Huang, Q. Gao, J. Liu, and D. Panda, "High performance virtual machine migration with RDMA over modern interconnects," in *Proceedings of the 9th IEEE International Conference on Cluster Computing*, 2007, pp. 11–20.

[105] B. Urgaonkar and P. Shenoy, "Sharc: Managing CPU and network bandwidth in shared clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 1, pp. 2–17, 2004.

[106] 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*, USA, 2006, pp. 199–212.

[107] 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)*, 2000, pp. 90–101.

[108] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," in *ACM SIGOPS Operating Systems Review*, vol. 35 (5), 2001, pp. 103–116.

[109] L. He, S. Jarvis, D. Spooner, X. Chen, and G. 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*, 2004, pp. 402–409.

[110] H. Lee, D. Lee, and R. Ramakrishna, "An enhanced grid scheduling with job priority and equitable interval job distribution," *Advances in Grid and Pervasive Computing*, pp. 53–62, 2006.

[111] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@ home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[112] D. Anderson, "BOINC: A system for public-resource computing and storage," in *Proceedings of 5th IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 4–10.

[113] B. Richard and P. Augerat, "I-cluster: Intense computing with untapped resources," in *Proceedings of 4th International Conference on Massively Parallel Computing Systems*, 2002, pp. 127–140.

[114] 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*, 2002, pp. 141–148.

[115] M. Salehi, H. Deldari, and B. Dorri, "Balancing load in a computational grid applying adaptive, intelligent colonies of ants," *Informatica: An International Journal of Computing and Informatics*, vol. 33, no. 2, pp. 151–159, 2009.

[116] C. Dumitrescu, I. Raicu, and I. Foster, "Di-gruber: A distributed approach to grid resource brokering," in *Proceedings of ACM/IEEE Conference on Supercomputing*, 2005, pp. 38–45.

[117] 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*, vol. 67, no. 8, pp. 957–966, 2007.

[118] R. Buyya, R. Ranjan, and R. 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*, 2010, pp. 13–31.

[119] R. Buyya, C. 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*, vol. 25, no. 6, pp. 599–616, 2009.

[120] 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 (IEEE CLOUD '10)*, 2010, pp. 236–243.

[121] H. Van, F. Tran, and J. Menaud, "SLA-Aware Virtual Resource Management for Cloud Infrastructures," in *Proceedings of the 9th Conference on Computer and Information Technology- Volum 02*, 2009, pp. 357–362.

[122] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman *et al.*, "Reservoir-when one cloud is not enough," *Computer Journal*, vol. 44, no. 3, pp. 44–51, 2011.

[123] A. Toosi, R. Thulasiram, and R. Buyya, "Financial option market model for federated cloud environments," Department of Computing and Information System, Melbourne University, Tech. Rep., 2012.

[124] A. Toosi, R. Calheiros, R. 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 '11)*, 2011, pp. 279–287.

[125] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle, "Dynamic Virtual Clusters in a Grid Site Manager," in *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, USA, 2003, pp. 90–98.

[126] R. Buyya, M. M. Murshed, D. Abramson, and S. Venugopal, "Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm," *Software: Practice and Experience*, vol. 35, no. 5, pp. 491–512, 2005.

[127] A. d. Costanzo, C. Jin, C. A. Varela, and R. Buyya, "Enabling Computational Steering with an Asynchronous-Iterative Computation Framework," in *Proceedings of the 5th IEEE International Conference on e-Science*, USA, 2009, pp. 255–262.

[128] N. Fallenbeck, H. Picht, M. Smith, and B. Freisleben, "Xen and the art of cluster scheduling," in *Proceedings of the 1st International Workshop on Virtualization Technology in Distributed Computing (VTDC '06)*, 2006, pp. 4–12.

[129] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, "A toolkit for modelling and simulating data grids: an extension to GridSim," *Concurrency and Computation: Practice and Experience (CCPE)*, vol. 20, no. 13, pp. 1591–1609, 2008.

[130] U. Lublin and D. Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 63, no. 11, pp. 1105–1122, 2003.

[131] A. D. Costanzo, M. D. Assuncao, and R. Buyya, "Harnessing cloud technologies for a virtualized distributed computing infrastructure," *IEEE Internet Computing*, vol. 13, no. 5, pp. 24–33, 2009.

[132] J. Anselmi and B. Gaujal, "Optimal Routing in Parallel, non-Observable Queues and the Price of Anarchy Revisited," in *Proceedings of the 22nd International Teletraffic Congress (ITC '10)*, Netherlands, 2010.

[133] L. Kleinrock, *Queueing systems: Computer applications.* Wiley-interscience, 1976.

[134] W. Tang, N. Desai, D. Buettner, and Z. Lan, "Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P," in *IEEE International Symposium on Parallel and Distributed Processing (IPDPS'10)*, 2009, pp. 1–11.

[135] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan, "Scheduling of Parallel Jobs in a Heterogeneous Multi-site Environment," in *Job Scheduling Strategies for Parallel Processing (JSSPP '03)*, 2003, pp. 87–104.

[136] L. He, S. Jarvis, D. Spooner, H. Jiang, D. Dillenberger, and G. Nudd, "Allocating non-real-time and soft real-time jobs in Multi-Clusters," *IEEE transactions on Parallel and Distributed Systems (TPDS)*, pp. 99–112, 2006.

[137] M. Colajanni, P. Yu, and V. Cardellini, "Dynamic load balancing in geographically distributed heterogeneous web servers," in *Proceedings of the 18th International Conference on Distributed Computing Systems*, 2002, pp. 295–302.

[138] S. Zhou, "A trace-driven simulation study of dynamic load balancing," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1327–1341, 2002.

[139] A. Hordijk and D. V. Laan, "Periodic routing to parallel queues and billiard sequences," *Mathematical Methods of Operations Research*, vol. 59, pp. 173–192, 2004.

[140] C. Grimme, J. Lepping, and A. Papaspyrou, "Prospects of collaboration between compute providers by means of job interchange," in *Job Scheduling Strategies for Parallel Processing (JSSPP '08)*, 2008, pp. 132–151.

[141] M. A. Murphy, B. Kagey, M. Fenn, and S. Goasguen, "Dynamic Provisioning of Virtual Organization Clusters," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, 2009, pp. 364–371.

[142] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 22, no. 6, pp. 931–945, 2011.

[143] L. He, S. Jarvis, D. Spooner, X. Chen, and G. 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*, 2004, pp. 402–409.

[144] J. Almeida, V. Almeida, D. Ardagna, i. Cunha, C. Francalanci, and M. Trubian, "Joint admission control and resource allocation in virtualized servers," *Journal of Parallel and Distributed Computing*, vol. 70, pp. 344–362, April 2010.

[145] M. Islam, P. Balaji, P. Sadayappan, and D. Panda, "Qops: A QoS based scheme for parallel job scheduling," in *Proceedings of 9th International Workshop on Job Scheduling Strategies for Parallel Processing*, 2003, pp. 252–268.

[146] S. Bose, *An introduction to queueing systems.* Springer, USA, 2001.

[147] D. Irwin, L. Grit, and J. Chase, "Balancing risk and reward in a market-based task service," in *Proceedings of the 13th IEEE International Symposium on High performance Distributed Computing (HPDC '04)*, 2004, pp. 160 – 169.

[148] S. Sharifian, S. Motamedi, and M. Akbari, "A content-based load balancing algorithm with admission control for cluster web servers," *Future Generation Computer Systems*, vol. 24, no. 8, pp. 775–787, 2008.

[149] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2008.

[150] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The case for power management in web servers," *Power aware computing*, pp. 261–273, 2002.

[151] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, 2011.

[152] H. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. Rumble, E. De Lara, M. Brudno, and M. Satyanarayanan, "Snowflock: rapid virtual machine cloning for cloud computing," in *Proceedings of the 4th ACM European conference on Computer systems*, 2009, pp. 1–12.

[153] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[154] L. Liu, H. Wang, X. Liu, X. Jin, W. He, Q. Wang, and Y. Chen, "GreenCloud: a new architecture for green data center," in *Proceedings of the 6th International Conference on Autonomic Computing and Communications*, 2009, pp. 29–38.

[155] J. Kephart, H. Chan, R. Das, D. Levine, G. Tesauro, F. Rawson, and C. Lefurgy, "Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs," in *Proceedings of the 4th International Conference on Autonomic Computing*, 2007, pp. 24–32.

[156] M. Hellmann, "Fuzzy logic introduction," Laboratoire Antennes Radar Telecom, Equipe Radar Polarimetrie, Universite de Rennes 1, France., Tech. Rep., 2001.

[157] J. Lagorse, M. Simões, and A. Miraoui, "A multiagent fuzzy-logic-based energy management of hybrid systems," *IEEE Transactions on Industry Applications*, vol. 45, no. 6, pp. 2123–2129, 2009.

[158] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "Autonomic resource management in virtualized data centers using fuzzy logic-based approaches," *Cluster Computing*, vol. 11, no. 3, pp. 213–227, 2008.

[159] R. E. Precup and H. Hellendoorn, "A survey on industrial applications of fuzzy control," *Computers in Industry*, vol. 62, no. 3, pp. 213–226, 2011.

[160] J. Jang, "Anfis: Adaptive-network-based fuzzy inference system," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 3, pp. 665–685, 1993.

[161] A. Toosi and M. Kahani, "A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers," *Computer Communications*, vol. 30, no. 10, pp. 2201–2212, 2007.

[162] L. Wang, *Adaptive fuzzy systems and Control Design and Stability Analysis.* Prentice Hall, 1994.

[163] "Parallel workloads archive," http://www.cs.huji.ac.il/labs/parallel/workload/.

[164] B. Sotomayor, R. Montero, I. Llorente, and I. Foster, "Capacity leasing in cloud systems using the opennebula engine," in *Workshop on Cloud Computing and its Applications*, vol. 1, 2008.

[165] D. Hitz, J. Lau, and M. Malcolm, "File system design for an nfs file server appliance," in *Proceedings of the USENIX Technical Conference.* USENIX Association, 1994, pp. 235–246.

[166] R. Noronha and D. K. Panda, "IMCa: A High Performance Caching Front-End for GlusterFS on InfiniBand," in *Proceedings of the 37th International Conference on Parallel Processing (ICPP '08)*, USA, 2008.

[167] S. Garg, "Meta scheduling for market-oriented grid and utility computing," Ph.D. dissertation, University of Melbourne, Department of Computer Science and Software Engineering, 2010.

[168] L. Barsanti and A. Sodan, "Adaptive Job Scheduling Via Predictive Job Resource Allocation," in *Job Scheduling Strategies for Parallel Processing (JSSPP '07)*, 2007, pp. 115–140.

[169] G. Utrera, J. Corbalan, and J. Labarta, "Implementing malleability on mpi jobs," in *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques (PACT '04)*, USA, 2004, pp. 215–224.

[170] A. Bucur and D. Epema, "Priorities among multiple queues for processor co-allocation in multicluster systems," in *Proceedings of the 36th Annual Symposium on Simulation*, 2003, pp. 15–23.