10-17-2012

Interoperable Resource Brokering with Policy-based Provisioning and Job Allocation

David Villegas
Florida International University, dville00@gmail.com

DOI: 10.25148/etd.FI12120515

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

INTEROPERABLE RESOURCE BROKERING WITH POLICY-BASED PROVISIONING AND JOB ALLOCATION

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

David Villegas

2012

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by David Villegas, and entitled Interoperable Resource Brokering with Policy-based Provisioning and Job Allocation, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Ming Zhao

Jason Liu

Jorge Rodríguez

Seyed Masoud Sadjadi, Major Professor

Date of Defense: October 17, 2012

The dissertation of David Villegas is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2012

DEDICATION

To my parents.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Masoud Sadjadi, for his support during my time at FIU. Also, I must acknowledge the important role of all my committee member, Drs. Ming Zhao, Jason Liu and Jorge Rodríguez. Their feedback during the proposal defense and afterwards has helped to greatly improve the quality of this work. I am also indebted to our collaborators at IBM Research, Liana Fong, Norman Bobroff and Yanbin Liu, as well as Dr. Ivan Rodero, from Rutgers University. They were invaluable from the first days of my PhD until the end, by giving their advice and helping direct my research efforts.

Much of the work here initiated at T.U. Delft, The Netherlands, under the guidance of Dr. Alexandru Iosup. I am thankful to him, Nassos Antoniou and Dr. Dick Epema for the very fruitful time spent there. I must also thank Drs. Ignacio Llorente, Rubén Montero and the OpenNebula team for giving me the opportunity to work with them.

I am grateful to those who were part of my academic life at FIU and helped by listening to my ideas and giving their feedback, among them my labmates Selim Kalayci, Dr. Javier Delgado, Xabriel Collazo-Mojica, Mani Shafa'at Doost and Karl Morris. Eric Johnson and his team were indispensable in helping manage the infrastructure used in the experiments. Dr. Geoffrey Smith provided the LaTeX template. Finally, I would like to thank my family for the continued support through all the years, and in special my beloved wife, Yina Hurtado, for his encouragement and understanding. Nothing of this would have been possible without her.

ABSTRACT OF THE DISSERTATION

INTEROPERABLE RESOURCE BROKERING WITH POLICY-BASED PROVISIONING AND JOB ALLOCATION

by

David Villegas

Florida International University, 2012

Miami, Florida

Professor Seyed Masoud Sadjadi, Major Professor

The increasing needs for computational power in areas such as weather simulation, genomics or Internet applications have led to sharing of geographically distributed and heterogeneous resources from commercial data centers and scientific institutions. Research in the areas of utility, grid and cloud computing, together with improvements in network and hardware virtualization has resulted in methods to locate and use resources to rapidly provision virtual environments in a flexible manner, while lowering costs for consumers and providers.

However, there is still a lack of methodologies to enable efficient and seamless sharing of resources among institutions. In this work, we concentrate in the problem of executing parallel scientific applications across distributed resources belonging to separate organizations. Our approach can be divided in three main points. First, we define and implement an interoperable grid protocol to distribute job workloads among partners with different middleware and execution resources. Second, we research and implement different policies for virtual resource provisioning and job-to-resource allocation, taking advantage of their cooperation to improve execution cost and performance. Third, we explore the consequences of on-demand provisioning and allocation in the problem of site-selection for the execution of parallel workloads, and propose new strategies to reduce job slowdown and overall cost.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1

INTRODUCTION

The increasing needs for computational power are evident when looking at areas such as weather simulation, genomics, computer-aided drug design or particle physics. These needs, coupled with the commoditization of high-performance computing infrastructure, have led to a proliferation of data centers for providing computing cycles. In some cases, such systems are maintained and consumed in-house; in others, dedicated providers lease their usage based on some contract, often called Service Level Agreement or SLA. A third, and increasingly popular alternative, is for institutions with available resources to share unused cycles, temporarily acting as providers, in order to maximize utilization and amortizing their ownership cost.

Multiple research efforts have been geared towards creating effective methods to allow the execution of workloads on shared resources among institutions. There are a number of challenges to be solved in order to provide seamless distribution of computing power between partners. First, standardized protocols need to be devised to authorize users across institutional boundaries, exchange resource information or execute user processes on remote machines. Then, scheduling policies have to be used by consumers and providers in order to optimize certain goals, such as execution performance or usage cost. Finally, collaborating sites need to implement effective placement strategies to exchange workloads among them.

It can be argued that the desired goal for computation sharing is akin to that already achieved by traditional utilities such as water or electricity. The term of utility computing [Rap04] was coined after such idea. According to this model, computational resources would be traded based on supply and demand rules, and consumption balanced as in the case of power generation facilities. One particular solution towards this goal, grid computing [FKT01], gained significant attention in recent years by the academic community, especially in the field of High Performance Computing (HPC). The grid model, however, suffers from a number of shortcomings. First, resources are under complete control of providers, resulting in a lack of flexibility for users requiring particular libraries or operating systems. Additionally, new mechanisms are needed to orchestrate the sharing when the number of resources is increased.

More recently, a new trend has appeared, known as cloud computing [AFG$^+$09], which borrows many of the ideas generated in grid and utility computing research. This new approach solves some of the existing problems by using new advances in virtualization and introducing better cost measures and billing facilities. Virtualization allows providers to better partition and isolate their resources and users to have higher control of their workloads. However, clouds lack standardized interoperability mechanisms to enable sharing of resources between providers, fine-grained control of resource performance and coordination among leased resources. Additionally, there are no well-defined methods to map job workloads to acquired resources.

1.1 Research Problem

Scientists communicate to execution sites, or providers, through the *job* abstraction. Computational jobs represent required calculations, and are usually defined by users themselves. A job specifies which program to run, a set of requirements in terms of computer architecture, Operating System or libraries, necessary input and output files, and the desired degree of parallelism —*i.e.,* the number of processors to be used during execution.

Users, through submitted jobs, acquire control of a certain number of resources for a limited amount of time. Varying goals such as cost or response time minimization can be specified as part of the job's definition. Providers need to be able to respond to such demands while maintaining their own goals in terms of utilization, power consumption or infrastructure costs, adapting to different request patterns produced by seasonal spikes, unexpected failures or demand increase.

There have been various attempts to solve these problems in various research areas, and technologies have been devised to allow users and providers to attain their goals. However, there are still multiple challenges that need to be undertaken to provide common protocols to publish and consume computational resources in a scalable fashion, while improving overall usage and reducing cost.

Grid computing was perhaps the first widely embraced technology for resource sharing across organizational boundaries. However, a lack of flexibility has forced users to learn intricate tools and protocols in order to be able to use them, restraining advanced usage in which complex QoS requirements are necessary. Additionally, provider sites employ batch schedulers on physical resources, resulting in a lack of control over the executing fabric, libraries or operating systems.

The more recent technology of cloud computing addresses some of the grid's flexibility problems by taking advantage of virtualization and provides more advanced billing mechanisms, but still lacks crucial features. First, no standardized federation protocols and methods for sharing user load have been defined, therefore leaving the problem of delegating excess requests to external providers to each site. This has led to ad-hoc solutions that can't be reused.

Finally, there is a lack of mechanisms to coordinate the achievements of cluster and grid computing in the areas of job scheduling and sharing with the new capabilities of clouds in terms of virtual resource provisioning. This is a critical step towards ensuring that user workloads can be executed while harnessing the flexibility of the new platforms and adequately mapping domain requirements to resource requests.

Therefore, we identify the general problem as the need to seamlessly execute parallel scientific workloads with different requirements among collaborating organizations. Such organizations may employ different protocols and middleware, provide heterogeneous resources —physical and virtual— and implement distinct local job scheduling policies in the case of physical resources, or virtual machine acquisition in the case of clouds. Thus, workloads need to be distributed in an interoperable fashion and resources have to be acquired across organizational boundaries by using mechanisms to abstract heterogeneity and improve certain metrics such as utilization or job slowdown. Finally, workload tasks have to be mapped to the acquired resources using strategies to optimize user and site objectives such as runtime, throughput, performance or cost. All these problems need to be solved in a coordinated manner so that low-level details are hidden from users.

We define the following research problems for this dissertation:

- **Heterogeneity in resource management mechanisms across organizations.** There is a need for interoperable mechanisms between providers to offload resource utilization while maintaining both consumers' goals and their own goals in a scalable fashion and conserving site independence in terms of infrastructure, policies and internal implementations. An organization using the Globus middleware [FK96] should be able to share its infrastructure with another peer employing UNICORE [ES01] or its own local resource manager such as Condor [TTL05] in order to balance the incoming load, without requiring any changes in their internal implementations.
- **Lack of understanding about the interplay of job scheduling policies and resource provisioning policies.** Strategies to provision virtual infrastructure and to schedule jobs to resources are not usually coordinated, and current efforts to provide an integration between them address the problem in an ad-hoc manner, which makes existing solutions only applicable to con-

crete cases. An integration of these two stages is necessary in order to achieve support for heterogeneous workload execution while fulfilling requested goals such as makespan, cost or utilization. For example, widely used job scheduling policies such as Backfilling or First Come First Serve that are unaware of how resources are provisioned may result in infrastructure waste and higher job slowdown due to the overhead of spawning virtual resources on-demand.

- **Lack of site-selection mechanisms for virtual resource providers.** During the site-selection phase, existing brokers are constrained to limited information and control when collaborating sites provide physical resources, accessed by batch schedulers. With the rapid embrace of virtualized infrastructure by scientific users, this problem needs to be revised to provide better decisions that improve utilization across organizations. Due to the control that virtual machines provide users in terms of provisioning and the ability to implement custom scheduling strategies, new site-selection policies need to be implemented to take advantage of the new capabilities.

Therefore, we enunciate the following research objective:

> **The goal of this dissertation is to improve the existing scheduling mechanisms to execute scientific parallel workloads among collaborating institutions in a de-centralized, scalable and agnostic manner. New opportunities brought by virtualization both in terms of resource provisioning and scheduling control need to be considered, as well as the role of this new paradigm in the site-selection process performed by job brokers or meta-schedulers.**

In this work, we address some of the mentioned problems hoping to advance the state of the art towards a more flexible and seamless method to execute computational workloads on shared resources. In particular, we concentrate in three main aspects. First, we investigate a grid interoperability protocol to overcome providers' heterogeneity in terms of middleware and execution resources; we develop a prototype and validate it by performing experiments across three different organizations. Second, we identify the two phases involved in scheduling job workloads to virtual resources; we study different policies for these two phases and propose new strategies that take advantage of their collaboration to improve workload metrics such as cost or job slowdown. Then, we combine our findings from the previous two areas to articulate a novel meta-scheduling model to execute parallel workloads across separate virtual providers. Meta-schedulers can take advantage of the available information in virtual resource provisioning and job allocation at other sites to make better decisions.

The rest of this dissertation is organized as follows: In Chapter 2 we give a general overview of the areas involved in this work and introduce the definitions used throughout the text. In Chapter 3 we describe existing work related to ours, and discuss similarities and differences between them. In Chapter 4 we introduce our findings in the area of grid meta-scheduling to share computational jobs among institutions. Chapter 5 motivates the transition from grid meta-scheduling to collaboration between virtual providers at different levels. Chapter 6 contains a study and definition of scheduling policies to map jobs to resources and provision virtual execution environments. In Chapter 7 we explore a new meta-scheduling model for virtual resource providers that results in better placement decisions. Finally, we summarize our work and describe the remaining steps in Chapter 8.

CHAPTER 2

BACKGROUND

Computationally expensive problems have arisen in many scientific areas, and consequently, increasing efforts have been directed towards harnessing more power from computers. We are specially interested in those initiatives directed towards creating distributed processing systems, this is, parallelizing computational demands and orchestrating their execution on multiple resources. In this section we take a look at existing technologies that address such problems from different perspectives, their advances in the field and also their shortcomings.

We first consider clusters of commodity machines, and then discuss two other technologies, grids and clouds, which address the challenges of complexity and large scale by applying ideas from the field of utility computing [Rap04, EAB^{+}04]. The main characteristics of this model, which has had an important impact in both of these areas, can be enumerated as follows:

- *Standardization:* Access to resources using a high-level, agreed-upon abstraction that hides the underlying complexity and heterogeneity.
- *Elasticity:* The ability to acquire resources on-demand with different granularities and depending on the usage needs.
- *Scalability:* Resource utilization can be increased and decreased quickly and seamlessly to satisfy changes in demand.
- *Reliability:* The need of having mechanisms to control and react to underlying failures in order to provide certain security to the users.

2.1 Cluster Computing

Due to the decreasing price of commodity computers and the higher speeds of the interconnection fabric, distributed computing has received increasing attention from the HPC community. Early experience reports [CCF+94] conclude that "cluster computing is a legitimate and viable computing environment for compute-intensive work".

Initial attempts to provide a parallel, distributed architecture for problem solving include the development of programming environments for process execution and intercommunication such as PVM [Sun90] or MPI [For93], execution infrastructures such as Beowulf [SBS+95], MOSIX [BL98] or NOW [ACP95], and local resource management systems for scheduling jobs among distributed resources such as Condor [LLM88] or PBS [Hen95]. The first group comprehends frameworks directed to developers to implement programs capable of executing and communicating over networked machines, providing programming constructs for message passing and synchronization.

Different organizations also tackled the problem of how to interconnect commodity hardware though high speed networks. In the case of Beowulf, a research team from NASA joined 16 processors which surpassed 1 Gigaflop for a real world application, using a easily-replicable system using freely available software and off-the-shelf components. This approach, known as "Pile-of-PCs" has been reproduced ever since by industry and research institutions. Baran and La'adan took a different approach with MOSIX by modifying the BSD Operating System to enable multi-user, time-shared process execution among interconnected resources with support for migration and load-balancing.

Finally, Local Resource Management Systems (LRMSs) such as Condor, PBS or Sun Grid Engine have been developed for multiplexing physical resources among users in a particular organization. Different scheduling algorithms have been researched to achieve various goals such as high throughput, fair sharing of resources or use of idle machines.

2.2 Grid Computing

Cluster computing has served as a useful paradigm for many years. However, with the continuing increase of resource requirements, there was a need to scale the computing infrastructure beyond the limits of an institution. Therefore, new techniques to share resources between heterogeneous, geographically separated organizations were required. In [CFK$^+$98], Czajkowski et al. describe five challenges in resource management for these cases, which they name *metacomputing* environments:

- *Site autonomy*, or the diversity between resources belonging to different organizations.
- *Heterogeneous substrate*, or differences in LRMS systems between sites.
- *Policy extensibility*, differences between applications and the need for sites to support them without knowing them *a priori*.
- *Co-allocation*, the requirement of executing parallel applications simultaneously at resources belonging to different organizations.
- *Online control*, or the negotiation of the application requirements based on dynamic resource availability.

Different systems have been developed by various groups in order to address these requirements. The Globus toolkit [FK96], precursor of the grid, is arguably one of the attempts that achieved more traction. It provides communication, authentication, network information and data access modules to enable the creation of such *metacomputing* environments. Other projects to unite remote administrative domains are Legion [GFKH99], which provides an object model so that new inter-operating components can be developed to communicate across sites, PUNCH [KF99], a web-based architecture to securely execute unmodified programs in remote resources through the use of a browser, DISCOVER [MP02], a distributed, peer-to-peer middleware substrate to allow services to interoperate which provides authorization, discovery, access control and coordinated interaction, or UNICORE [ES01], a middleware to execute Abstract Job Objects (AJOs), or Java programs defining user executions, to remote resources, providing security and monitoring capabilities.

These solutions aim at providing a common middleware for institutions to share their available resources. However, it is up to users to implement the algorithms to choose sites based on their requirements or preferences. In many cases, the user is in charge of making such decisions, leading to inefficient use of multi-domain resources. Thus, a component called the *meta-scheduler* is placed on top of the middleware to find and request the appropriate resources on behalf of the user.

2.3 Cloud Computing

Cloud computing represents an evolution of the grid towards a more flexible resource model where virtualization plays a central role. Multiple researchers have tried to define and categorize clouds ([AFG$^+$09, YBDS08]), and given the popularity of the paradigm, there have been different and even contradicting definitions used by industry and researchers.

There are some basic assumptions about cloud behavior, however, and they closely relate to utility computing attributes: cloud providers need to be flexible in adapting to user demand, implementing mechanisms to scale up and down the amount of provisioned resources, which can be acquired on-demand.

Attempts to categorize clouds have resulted in various taxonomies. One possible distinction identifies three main layers of operation: Software as a Service (SaaS), the layer where applications exist, which serves high-level, domain-specific workloads; Platform as a Service (PaaS), the intermediate layer where middleware, software licenses or programming stacks reside, and Infrastructure as a Service (IaaS), the lowest layer where virtual resources are managed. Examples of SaaS are services such as Google's Gmail [Gma], Salesforce [Sal] or research portals such as Teragrid Science Gateways [WDGK+08]. PaaS solutions are also found in industry (Microsoft Azure [Azu], Google AppEngine [GAE]) or academia (Aneka [CNJ+07]). Examples of IaaS are Amazon's Elastic Compute Cloud or EC2 [EC2] and Rackspace [Rac] in industry or OpenNebula [Ope], Eucalyptus [NWG+09] or Nimbus [KFFZ05] in academia.

A different categorization groups clouds according to their users: Public clouds offer service to external customers, who pay for the amount of resources used, private clouds are internal to an organization, and hybrid clouds are private clouds capable of acquiring external public resources based on user demand or other internal policies.

The main difference between clouds and grids is the lack of resource sharing mechanisms [VRF+10, FZRL08]. In fact, most of the solutions that initially appeared labeled as "cloud" only consider resources in one organization. Recent efforts such as RESERVOIR [RBL+09] have tried to define federation protocols. We explore them in Section 3.1.2.

CHAPTER 3
LITERATURE REVIEW

In this chapter we describe existing work in the literature which is related to ours, and therefore needs to be considered in order to understand the approach described here. Section 3.1 addresses interoperability between resource providers by examining advances in grid meta-scheduling, cloud federation and network testbeds. In section 3.2, we list related work in the area of virtual resource management, both for execution environments and network resources. Finally, section 3.3 discusses existing policies and algorithms used in virtual resource provisioning for job execution.

3.1 Site Interoperability

When resources are shared by different organizations, there are a number of challenges to be solved in order to orchestrate either the execution of user workloads among them or to acquire the infrastructure in a seamless manner. Usually, heterogeneity needs to be abstracted by implementing standardized protocols and using a common resource description model. Then, selection algorithms that account for infrastructure characteristics need to be run, and finally, either jobs need to be mapped and executed on remote resources, or techniques to abstract geographic separation and resource heterogeneity need to be employed to present the end user with a consolidated substrate. In this section we examine efforts towards these goals in the fields of grid computing, clouds and network testbeds.

3.1.1 Grid Meta-Scheduling

A grid meta-scheduler or grid broker [KBM02, HML10] is the component in charge of processing user requests for jobs with associated resource requirements, querying available sites, selecting the most appropriate ones based on some pre-defined policies, and submitting the request following the required protocol [Sch04]. The meta-scheduler component is in charge of bridging the gap between organizations and hide the complexity and heterogeneity of underlying resources, administration policies and middleware.

The AppLeS metacomputing scheduler [BW97] represents an early system directed towards efficiently scheduling applications in heterogeneous and shared resources. It addresses the problem by analyzing the application's needs, the resources' state and availability at a given time and predicting their future state to determine the best schedule. Many ideas are common with our layered approach to federation, especially the need to transfer user and application requirements from the top layer to the infrastructure, although the use case scenario has changed due to the addition of virtual resources and organization independence. We attempt to solve them by considering inter-site protocols and different job to virtual resource mapping policies.

Vadhiyar et al. [VD02] also define a meta-scheduling architecture where application level considerations are taken to determine the best schedules. The difference with AppLeS is that in this case more advanced negotiation policies are implemented to determine whether an application should run when requested, or wait to improve the overall performance of all applications. This is a characteristic that is used in more recent work to improve global execution of applications, including ours.

The GrADS [DBC03] project proposes a centralized scheduler capable of assigning user tasks to remote resources, given that the job submitter specifies a performance model and a task-to-processor mapping. Their approach consists in exhaustively searching available resources with pruning of candidates that don't meet the mapping requirements. This algorithm reduces the search space and provides good results. We employ a similar approach when trying to schedule requests to a particular site.

Sabin et al. [SKR03], explore meta-scheduling of parallel jobs in heterogeneous resources. Assuming scheduling strategies at each site can be controlled, they evaluate different backfilling scheduling methods taking site heterogeneity into account. We share the premise of heterogeneous resources and different types of workloads, such as parallel, and also study similar policies. However, we separate the mapping of jobs to virtual resources from the resource provisioning, and allow different algorithms for these two phases.

Ramakrishnan et al. [RIG+06] discuss a model where different grids share resources by defining two separate levels of abstraction, namely jobs and containers. Here grids are hosted by different sites and a component, called the *GROC* is in charge of routing tasks between sites and determining whether to increase or decrease the number of execution nodes for a particular VO. This work is similar to ours in the separation between jobs and resources. However, there are two fundamental differences: first, the GROC component is centralized, and one instance determines the task flow for all users of a certain grid. Second, different GROC instances cannot collaborate among them, but rather sharing is only done at the resource level. This does not allow virtual resources controlled by different GROCs to be shared among users from different VOs, and can therefore lead to fragmentation and lack of container reuse.

The GridWay meta-scheduler [HML07] provides interoperability between different grids though adapters. It supports the JDSL language [ABD+05] and can perform matchmaking on the resources and implement pluggable scheduling algorithms. We build up on their idea by allowing multiple meta-scheduling components to collaborate so that resource competition is orchestrated at the highest level instead of the infrastructure level. By defining an inter-site negotiation protocol, different instances of GridWay are able to collaborate among them to provide better resource usage.

The KOALA grid scheduler [ME08] implements parallel task co-allocation across resources, and implements two policies to decide where to place jobs, *Close-to-Files* and *Worst-Fit*. It reserves processors in order to accomplish it, and implements a fall-back mechanism when such function is not available in the underlying infrastructure. In KOALA, there is one centralized scheduling agent, and only the cost of the initial file transfer is considered, making it unsuitable for tasks with high communication requirements. Also, only physical resources are taken into account.

Iosup et al. [ITF+08a] address the problem of resource selection for inter-operating grids. It has a similar goal to our approach to meta-scheduling, but instead of forwarding jobs to sites, here resources are delegated to job requesters. The authors assume that a method like Condor glide-in [TTL05] is used to join remote resources to the originator's pool. Site managers in this architecture can delegate requests for resources to other participants, creating delegation chains. The system supports different policies to trigger the delegation process through a load management algorithm, or the order in which other sites are contacted. Although this work is promising, it has the shortcoming of not considering network topology into account when performing resource delegation, thus making it unreliable for parallel workloads. Also, the authors concentrate in defining the high-level delegation protocol, but they don not implement it in a real system, therefore making it unclear whether this model would be feasible in a realistic deployment.

Assunção et al. propose Intergrid [DdAaBV08], a formulation of the existing challenges to create dynamic interoperating grids of resources. The authors identify many of the problems in federating separate sites based on changing needs, and propose different methods to overcome them. They address resource virtualization, peering agreements, market-based incentives and network aware placement of requests, although they don't have an implementation. Our work shares many of the requirements contemplated in Intergrid, for which we provide real use cases, implementations and measurements.

Leal et al. [LHL09] propose a decentralized model of grid meta-schedulers and define four algorithms to determine where tasks should be sent in order to reduce workload makespan and improve overall execution performance. Their work is similar to our approach of distributed domains that exchange workloads and resources; however, they assume homogeneous and independent jobs. This results in a model that doesn't consider different resources, workloads and fluctuations in the infrastructure.

Caminero et al. [CRCC11] define a peer-to-peer meta-scheduling framework for distributed administrative domains. Their resource selection algorithms takes inter-site bandwidth into account, and tries to route job requests to the most appropriate neighbor by calculating site indices that represent the available computational power and bandwidth. Their approach follows a peer-to-peer communication model as ours, although in our case we differentiate the job scheduling and resource scheduling process.

3.1.2 Cloud Federation

Researchers have recently started to propose new protocols to enable the sharing of virtual resources among sites. This is known as federation, and it involves different mechanisms —many of which were already envisioned and implemented in grids [VRF+10]. Given the appropriate protocols for resource sharing, the next logical step is to develop brokering systems to orchestrate the execution of user workloads in different domains that offer virtualized systems. The behavior of such systems has some similarities with previous grid meta-schedulers, (need to choose among different domains, translate user requirements or monitor request state), but also some differences, especially considering the fact that cloud providers offer malleable resources.

Sky computing [KTMF09] describes the use of ViNE overlay networks [TF06] among cloud providers to provide trusted communications and sharing of resources. This interoperability is implemented at the IaaS layer, using existing APIs, although the authors acknowledge the problems of constraining to existing protocols which lack expressivity —*e.g.,* the use of VMs with *"high bandwidth."* They discuss some solutions such as running benchmarks on the infrastructure before submitting workloads to have a good idea of their performance. Finally, they experiment with a Hadoop application across clusters to demonstrate the scalability of their system. This work has many similitudes with ours at the larger scale. Our goal is also to join resources across providers, with the difference that instead of using existing APIs we define a federation protocol that sites can implement. Also, we differentiate between federation at the infrastructure and platform layers, thus being able to extract application requirements and interact with non-virtualized systems. Finally, our use of virtual networks, although similar to ViNE, is also geared towards determining whether aggregate QoS can be met by providers, helping in the brokering stage.

RESERVOIR [RBL$^+$09, RBE$^+$11] is closer to our approach in that it defines an interoperable architecture between providers. In RESERVOIR, sites provision Virtual Execution Environments (VEEs) capable of running *Service Applications*. Services are described by a Service Manifest file, which defines the desired SLA parameters and how Services have to be run among VEEs, including, for example, the necessary VM images. Also, a range of requested VEEs is specified, which can change based on certain elasticity rules. Differently from this work, where federation happens through a VEE Manager (VEEM), in charge of retrieving service manifests and communicating with VEE hosts and other sites, we decouple these tasks among the platform and infrastructure layers, thus allowing federation to happen among sites that may not implement the whole federation stack. Other work by Hadas et al. [HGR09] adds the concept of networking in RESERVOIR federations through Virtual Application Network (VAN) and Virtual Internet Access (VIA) services. However, this is still in the earliest stages and does not describe how request mappings involving network QoS requirements are made, how traffic control is implemented to fulfill user requests, or how the networking aspect is managed by the federation protocol. In other aspects, there are many similarities with our work, for example in classifying VAN communications as intra-host, intra-site and inter-site, or the use of Site proxies (similar to our *site gateway* agents).

InterCloud [BRC10] is another approach to cloud federation to provide a mechanism to join VMs, services, storage or databases from different providers to support dynamic scaling of resources to handle demand variations. In this research, a *Cloud Exchange* component coordinates bids and negotiations between users and providers, applying market-oriented principles for resource acquisition. The authors perform some initial tests using simulation, but it is unclear whether this model is applicable in a real-world scenario due to some shortcomings in the evaluation, such as not considering VM boot time, QoS requirements or the scalability of the Cloud Exchange component for multiple sites.

Tordsson et al. [TMMVL12] propose a cloud broker to place virtual machines across different providers while optimizing certain criteria specified by the user. They use an integer programming formulation to achieve the best placement, considering user requests in terms of VM capacity and price. For their implementation, they create a Sun Grid Engine deployment where the master node runs on local resources and the worker nodes are instantiated in different clouds and joined through OpenVPN. Their application is the Embarrassingly Distributed benchmark from the NAS Grid Benchmarks [FVdW02]. This work shows an actual implementations of grid federation, however, it is very constrained to a certain type of application (embarrassingly parallel) and limited request description (CPU, memory and price). Clearly, the solution of using OpenVPN to join nodes from different clouds would not work for many applications. Our work addresses this problem by allowing the definition of network requirements and instantiating virtual resources accordingly. Furthermore, our solution allows multiple sites to collaborate by exchanging their load, rather than approaching the problem from a hierarchical point of view where one broker is in charge of communicating with all providers.

Finally, there have been some efforts towards creating interoperable protocols for IaaS providers. The Open Grid Forum (OGF) approach, the Open Cloud Computing Interface (OCCI [OCC]), defines a REST protocol to describe, manage and monitor cloud resources. It supports computing, networking and storage resources, although it still lacks QoS constraints such as bandwidth or latency. The standard is in development, although it has been initially implemented by some IaaS managers.

3.1.3 Network Testbeds

There has also been an effort spawning from the network community to reach the same point of resource sharing, however focusing on the network aspect. Such projects achieve this by virtualizing network resources for researchers to run controlled experiments. To have an idea of the similarities between this and previous work, it must be noted how some researchers have recently addressed these testbeds as “network clouds”.

While the HPC community has advanced towards better scheduling and sharing methods for computational jobs that primarily take into account processor, memory and disk, there is clearly a disconnect with the advances in the networking community, which while concentrated in the experimentation of network protocols, progressed towards large distributed systems where to run their experiments. Such network testbeds have in few cases taken ideas from the HPC community or applied them back to it. It is our goal in this section to study such efforts in the areas of network virtualization, federation architectures and protocols, and resource management techniques. These categories fall under the wide umbrella of the "control plane" in systems such as Emulab [WLS$^+$02] or PlanetLab [PACR03].

Netbed, a descendant of Emulab [WLS$^+$02], is one of the initial attempts in providing simulation and emulation in a distributed testbed for network experimentation. This project shares many of our goals, namely, automating the deployment of experiments so that scientists can reproduce executions easily by defining their requirements. However, Emulab has a centralized management architecture that could not be used for our case, since one of our premises requires different domains to maintain their administrative privileges. Another difference is how experiments are specified: Emulab users write *ns* scripts to define the network requirements, we provide a request model geared towards the execution of applications. Finally, there is a need to execute experiments when resources are provisioned: we need to implement a job execution layer that runs the desired workloads.

PlanetLab [PACR03] provides a distributed service for researchers and users to test and deploy new applications with specific network requirements. PlanetLab creates overlays to reproduce the requested network in terms of latency, congestion and failure rate. The main differences with our approach are first the targeted audience: we allocate resources in order to execute different types of applications and not to reproduce the network conditions. Secondly, in our case resources are not opportunistic, but dedicated data centers belonging to different organizations, thus having to deal with the aspect of sharing.

ModelNet [VYW+02] is a distributed emulation platform to allow scientists to create Internet-scale reproducible experiments. In ModelNet, applications send unmodified traffic through a central cluster to emulate user-defined network conditions. This project shares some ideas with the previous ones and also with our approach, although its scope is different and completely focused on repeatability of network experiments, rather than real applications.

Shirako [ICG+06] provides a distributed system for brokering and acquiring leases on networked resources. It defines different roles such as *guests*, or users of the infrastructure, *providers* and *brokers*, which maintain information about site resources. Guests define their requirements and request tickets from brokers, which in turn receive them from sites. Then tickets are redeemed to acquire resource leases. Shirako maintains certain similitudes with our approach in how resources are acquired, although in our case brokers can negotiate among them to determine the source of resources. Also, we separate the job brokering and resource brokering phases, thus allowing sharing to happen at these two levels.

One step towards the integration of network testbed technologies (GENI) and cloud computing is discussed by Baldine et al. [BXM+10], where the authors combine GENI's ORCA control framework with the Eucalyptus virtualization manager. In this work, providers of computing, storage or network resources delegate them to broker entities that use allocation policies and handle utilization tickets to users. The authors address two main problems: first, the need to co-allocate heterogeneous resources, and second, to join remote resources to form a unified group to fulfill user requests. They solve the first problem by creating a detailed description of resources using an OWL [MvH04] ontology capable of describing semantic network and computing capabilities. For the latter, they perform an operation they call *stitching*, where VLAN labels from different providers are negotiated. This work has some commonalities with our approach, for example in the use of brokers and network provisioning with cloud providers. However, our work is more focused on the resource management side, for example by defining the protocols to interoperate across sites, or studying different allocation policies, while this solution supports more detailed network description and operations. There are other issues the authors leave for future work, such as how to provide connectivity across slices and control the traffic among them, which we already address in our work.

3.2 Virtual Resource Managers

The advances in hardware virtualization promptly attracted the attention of scientists looking for a more flexible solution for resource utilization. The use of Virtual Machines (VMs) for the execution of computational jobs was proposed many years ago [FDF03]. However, providers need to manage the ever-growing number of VMs employed to run different user workloads. Section 3.2.1 describes multiple examples of virtualization management for computing resources

3.2.1 Virtualization of Computing Resources

Virtualization managers are in charge of provisioning virtual resources, in the form of VMs, and making them available for users so that they fulfill their requirements in terms of computational power and required software. Different research projects explore such managers, which handle VM images, user accounts, and partitioning the infrastructure to create isolated containers. OpenNebula [SMLF09], Eucalyptus [NWG+09] or Nimbus [KFFZ05] implement such basic functionality. However, these approaches can only instantiate individual VMs, and do not have the required mechanisms to coordinate complex deployments. Additionally, they cannot cross institutional boundaries, requiring other components to do this.

Haizea [SKF08] adds additional scheduling features on top of OpenNebula to support advanced reservations in clouds. It takes advantage of VM suspension, migration and resuming to preempt leases and manages VM image transfers in advance in order to achieve the requested instantiation time. The authors demonstrate how using a mixed workload of best-effort and advanced reservation requests they can improve the overall resource utilization. Our work only considers best-effort requests, but the Haizea scheduling mechanism could be implemented easily in our system to support these additional features since it is designed as an OpenNebula module, which we use as the underlying IaaS manager.

The Virtual Computing Lab (VCL) [SAH+09] is a cloud and cluster manager with the goal of providing on-demand resources for university students. It supports the loading of either VMWare VMs or bare-metal images on a common substrate for desktop or high-performance machines. A scheduling component manages the available images, reservations and user accounts and performs provisioning on demand. Our approach to provisioning is orthogonal to this project in many aspects, and includes additional features for inter-site deployment and complex infrastructure topologies, which could be used to transparently handle requests for desktop applications with best-effort requirements and ensembles of VMs for computational tasks with specific latency and bandwidth constrains. We are currently collaborating with NCSU towards this goal.

The CometCloud cloud engine [KP11] provides the capability to integrate multiple clouds and grids to execute workloads with different QoS requirements, enabling *cloudbursting*, or scaling out for resources when demand peaks. It employs a decentralized distributed hash table (DHT) to query resources. The main difference with our work is that CometCloud supports workload heterogeneity at the programming layer, thus requiring an expert to implement an adaptor for each supported application so that QoS can be met. Also, in CometCloud different resources are managed by scheduling agents which don't cooperate among them, thus when multiple organizations share resources, their agents don't negotiate but instead acquire the infrastructure separately. A final difference is the model of operation: CometCloud uses a pull mode, where resources request tasks from the scheduling agent, while in our case we use push to assign jobs to resources.

Although they do not exactly fall in the domain of virtual resource managers, many research solutions considered as utility computing used some measure of virtualization. In some cases this is not achieved through a hypervisor, but more through lightweight methods, at the cost of sacrificing isolation. Our approach shares multiple ides from this work.

The Océano [AFF$^+$01] manager for computing utilities is a prototype in the area of e-business applications capable of performing different operations to the hosting resources in order to satisfy user SLAs, such as throttling the number of requests, allocating or deallocating servers, and monitoring the state of the hosted domains. Similarly to our case, Océano takes the underlying network infrastructure into account and isolates traffic from different users using VLANs. There are certain differences with our work, however. In our approach to satisfy user requests we allow for multiple sites, and therefore need to perform brokering of available candidates. We also support other cases besides web applications, such as HPC or virtual labs, which have different communication and execution patterns and dictate the identification of the proper substrate to satisfy users' QoS requirements.

SODA [JX03] represents another approach towards hosting services over distributed resources. This project spawns virtual service nodes on physical hosts to run user application services, while maintaining some level of isolation and customization. It is similar to our work in that it allows users to specify the characteristics of their services in terms of CPU, memory, bandwidth or number of machines, but differently from our approach they must host service nodes in all the resources. Also, they do not contemplate multiple sites or latency requirements, which greatly impacts the placement algorithms.

3.3 Job Scheduling Policies in Virtual Resources

With the embrace of virtual resources for job execution, there has been a need to explore and update the algorithms used to assign tasks to virtual containers. Virtual machines can be instantiated by a fraction of the cost and much faster than physical hosts, therefore favoring dynamic policies that consider cost and performance tradeoffs. Many researchers have investigated virtual resources from public clouds as an extension to private data-centers.

ElasticSite [MKF10] proposes a resource manager built on top of Nimbus [KFFZ05] which uses different policies to extend a local cluster with IaaS resources based on job workloads. Three policies, *OnDemand*, *SteadyStream* and *Bursts*, are considered, and different metrics such as utilization or number of used VMs are calculated. Compared to our work in allocation policies, this research considers external virtual resources to meet local requirements, while we do not differentiate between the source of the resources. Another important distinction is that this work focuses on the resource acquisition process, but not in the job to resource allocation.

Assunção et al. [dAdCB09] study three allocation policies and different provisioning policies for extending the capacity of a local private infrastructure with public cloud resources. Their work concentrates on backfilling algorithms, which we haven't yet explored. In our study, we do not assume users have a good estimate of the jobs execution time, a condition that is necessary for this kind of policies. Another difference with our work is that all results are produced by simulation, rather than executing the policies on real cloud systems.

Kijsiponse et al. [KV10] describe a system where virtual resources from a Eucalyptus cloud are added to the Torque batch scheduler. Virtual machines are instantiated when there are no available physical resources and is directed by two different policies: the first creates the VM type that is most requested by jobs and the second monitors the demand proportion for VMs, starting those that have a higher demand rate. Jobs are allocated to VMs using a FIFO policy. As in other works, this one explores the scenario of extending physical resources with external clouds and the considered policies, specially those for job to VM allocation, are not very extensive since they only focus on the FIFO strategy.

Lu et al. [LJE$^+$10] address the problem of idle instances when provisioning VMs on demand to execute BLAST workloads by dynamically scaling in Microsoft's Azure cloud. They also consider the problem of load imbalance generated throughout the workload execution due to different task complexity. They use a simple allocation algorithm, but detect imbalance and perform job checkpointing and resource reconfiguration to address resource waste. This work has a different approach than ours by "correcting" the resource allocation based on the workload rather than proposing different policies. However, it is tailored for parameter sweeping type of applications only.

The work by Genaud et al. [GG11] is closer to ours in that it considers different algorithms to provision VMs and assign jobs to them. However, the policies they explore are different to ours, and they are not implemented in a real environment but simulated. The experiments do not consider aspects such as VM startup time, which in our experience have a great impact in the performance of the different policies.

CHAPTER 4

A META-SCHEDULER FOR THE GRID

Grid computing provides the necessary tools for multiple partners forming a Virtual Organization to share resources among them. Operations to communicate individual resource state, define and execute jobs across sites or authorize users, are implemented by middleware such as the Globus Toolkit. However, the decision of where to submit jobs is delegated to the user, which needs to consider very large numbers of resources distributed across different providers. The problem is exacerbated due to infrastructure heterogeneity and different grid middleware implementation by providers. A component, called the meta-scheduler, is in charge of matching user requests to available resources provided by the Virtual Organization members. In this chapter we define a meta-scheduler architecture combining hierarchical and peer-to-peer models to enable interoperation between partners. We propose a set of protocols to maintain communication sessions, exchange resource information in a scalable manner and transfer job workloads among sites. We validate our architecture's functionality by running empirical experiments across three different sites, including FIU, IBM and BSC, and finally we propose a resource emulation methodology to allow for larger scale experiments while using actual protocol implementations.

4.1 Introduction

Recent advances in cooperating grids, or interoperating VOs, include fulfilling job requests using data and computing resources distributed across multiple grids. This vision of cooperating grids further provides the opportunity for global optimizations of resource usage, reducing job execution cost and power consumption. The challenges to achieve this vision of interoperable, globally distributed VOs are well documented. They must be addressed by providing common interface descriptions, system models, and levels of abstraction to hide heterogeneity in the computing resources, security mechanisms, and job management policies of the collaborating organizations, as articulated in the paper by Rodero, et al. [RGC^{+}08a]. Several architectures have been proposed to achieve these goals, including SPA [OTG^{+}05], GridWay [HML04] and Koala [ME08].

This chapter describes an approach based on interoperating meta-schedulers (MS). The interoperable MS model supports the autonomy of organizations and presents a common external interface to partner domains as indicated in Figure 4.1. It shows a peer-to-peer collaborative environment constructed for the purpose of experimentation between three VO domains among three partnering institutions within the LA Grid Initiative [B^{+}07]: Florida International University (FIU), Barcelona Supercomputing Center (BSC) and IBM Research (IBM). MSs serve as the point of contact for end users submitting jobs to these sites as well as for cooperating VOs to exchange control messages.

To enable interoperation among MS, interfaces and functions are defined for the operations of job lifecycle management and the distribution of current state and availability of system-wide computing resources. When a user submits a job, the MS decides whether to execute it on local resources under its control or forward it to another MS either because there is not enough suitable local resources or because the remote MS is better suited to execute the job. Since the grid is a dynamic environment, good forwarding decisions require global distribution of resource state data to each MS domain. As the system of Figure 4.1 is scaled up, the volume of data exchanged becomes an important issue. Forwarding based on inaccurate data may lead to failures in job dispatching, while real time exchanges of complete resource state incur significant network and storage costs at the MS endpoints.

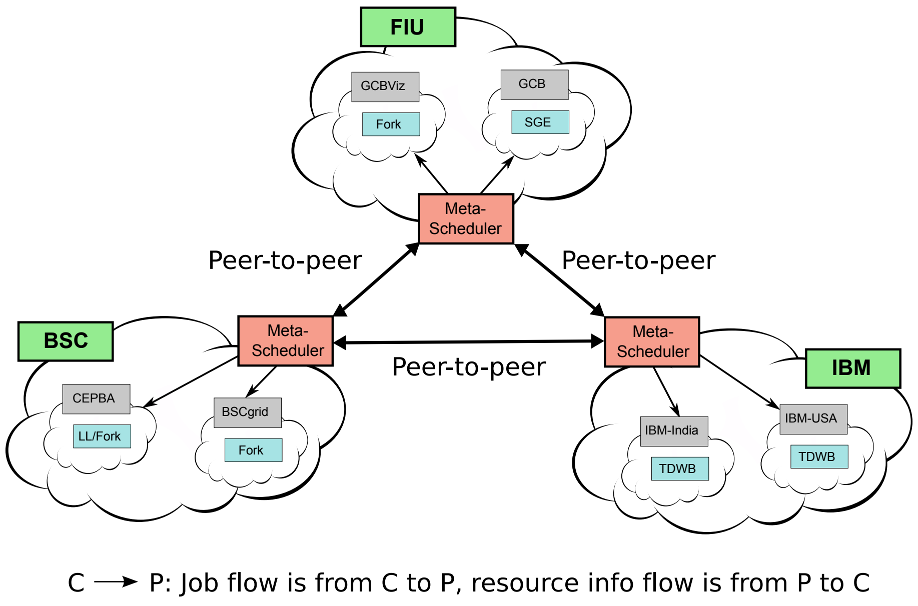


Figure 4.1: Cooperating meta-schedulers in LA Grid

A peer-to-peer —as opposed to centralized— model of resource data distribution is investigated to be consistent with our overall MS topology of Figure 4.1. In order to reduce the amount of data sent and stored at each node, several aggregation models are developed in Section 4.2.2. Each model represents an increasing degree of data consolidation, with the trade-off of decreasing the accuracy of state information.

The rest of this chapter is organized as follows: the MS model is described in Section 4.2. Section 4.3 shows how the common protocol is implemented for the FIU site. Section 4.4 demonstrates interoperability of three remote sites, providing performance data obtained running the high performance computing (HPC) Weather Research Forecasting (WRF) program. In section 4.5, we propose a resource emulation methodology to carry large scale experiments between meta-schedulers, while being able to measure differences in implementations. Section 4.6 concludes the chapter.

4.2 The Peer-to-Peer Model of Meta-Schedulers

The MS is the primary contact point for grid users and other MSs. Internally, MSs may have heterogeneous implementations, but adhere to a common set of communication protocols and information models that allow them to interoperate and provide a consistent view of the interconnected grids. The model adopted here is based on a peer-to-peer topology where an MS can interact with any other in a consistent way. There is no central or hierarchical structure such as a scheduling authority, global repository for resource information, or directory of MSs. Figure 4.1 shows such a model for the three interacting domains at BSC, FIU and IBM.

Flexibility is added to the peer-to-peer interactions through the introduction of *provider* and *consumer* roles. An MS in the provider role offers its resources for the execution of other MSs' jobs, while an MS with the consumer role requests other MSs' resources for the execution of its jobs. Providers advertise their sharable resources to connected consumers which in turn may propagate this information to their peers. When a job arrives to an MS, it decides whether to run the job locally within its domain or forward it to one of its providers. These roles are properties of the MS endpoints for a connection between MSs. An MS can be a provider to a set of MSs and simultaneously be a consumer to another set of MSs.

Assignment of one or both of these roles to the connection between two MSs allows administrators to shape the interconnected structure of the grid. The flow of resource information and forwarding of job submissions to providers is constrained by the provider and consumer roles of each connection. For example, in connecting the domains of BSC, FIU and IBM (Figure 4.1) each MS is both a provider and a consumer to other MSs. Thus, resource information fully distributed between partners and jobs can be submitted or forwarded to every domain. In contrast, a centralized model is created using a 'star' topology where a central MS is a consumer to all connected 'edge' MSs and edge MSs have both provider and consumer roles to the central MS. Since the star has no connections between edge MSs, only central MS will have the information of resources at every edge MS and be able to forward jobs.

The MS roles can be assigned and changed dynamically. For example, when a domain administrator makes available the domain's compute resources for sharing only during periods of low local activity. Here the domain's external facing MS is a consumer most of the time, while taking on both consumer and provider roles off shift. In this example, MSs internal to the administrative domain may be providers at all times.

Multiple capabilities and operations are required on the part of the MS to participate in the peer-to-peer model. Membership of the peer-to-peer network overlay requires each member to support the basic set of the inter-operation protocols described in the next sub-section.

4.2.1 Protocols and Interfaces

There are three types of operations between MSs as indicated in Figure 4.2 and Table 4.1: connection and communication, resource exchange, and job life cycle management. The connection protocol initiates membership in the grid and negotiates parameters including the authentication method and parameters, the assumed roles, the heartbeat rate to monitor connection status, and the resource and job description language understood by the parties. *openConn()* messages are exchanged, multiple rounds if necessary, between two MSs during the negotiation until an agreement is reached, or the number of rounds exceeds a threshold specified by the initiator, in which case the connection attempt fails. Once a connection is established, *heart-*

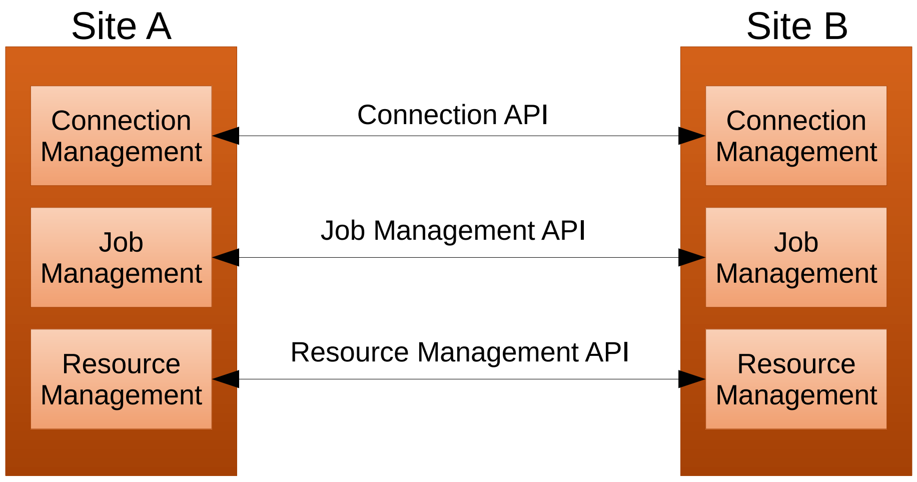


Figure 4.2: Meta-Scheduling protocol

Connection Messages	**Resource Information Messages**	**Job Execution Messages**
openConn()	*requestResourceData()*	*submitJob()*
notifyConn()	*sendResourceData()*	*queryJob()*
heartbeat()		*notifyJob()*
		cancelJob()

Table 4.1: List of messages in the LA Grid meta-scheduling protocol

beat() messages are exchanged using the negotiated intervals and the *notifyConn()* message is used to notify partners of error conditions or gracefully terminate the connection.

Resource exchange occurs between connected MSs using either pull mode (*requestResourceData()*) by a consumer, or push mode by a provider (*sendResourceData()*. The provider triggers *sendResourceData()* when the dynamic changes in resource capacity, utilization, or availability are over a threshold. Resource updates may be full or incremental. The consumer typically requests full updates in pull mode, while the provider generally pushes incremental updates.

Job requests are submitted using the *submitJob()* message. Our implementation uses JSDL (Job Submission Description Language [ABD+05]), an OGF[1] proposed recommendation, as the standard format for job submission. The receiving MS creates a local record of the submission and assigns a unique identifier to the job, which is returned to the submitter. Then it checks for a match against its resources and decides whether to schedule the job locally, or try to match the requirements to the available resources of other MSs. A job is forwarded between MSs using the same *submitJob()* interface and procedures. The job is tagged with a web service end point reference (EPR) of the forwarding MS. This process is repeated until the job reaches an MS that executes the job using its local resources. A map of forwarding EPRs is retained at each intermediate MS to allow job status updates to be sent back from the executing domain to the original submitter. The *notifyJob()* message is used to asynchronously inform the client of job status using this chain of forwarding MSs. The submitting client or a system administrator queries the job state or cancels the job through the synchronous *queryJob()* or asynchronous *cancelJob()* messages.

4.2.2 Resource Model

One of the main challenges arising from the large scale of grids is the need for efficient communication of available resources and their status. Partnering sites may have multiple heterogeneous execution infrastructure, which needs to be described to other peers in order to make decisions about job placement. Additionally, the peer-to-peer model presents the problem of "information aging" compared to centralized

[1]http://www.ogf.org

solutions. Concretely, data received from partners will loose validity with time, requiring constant updates to maintain its relevance.

Therefore, we define a resource model with enough flexibility to represent heterogeneous infrastructure, while supporting efficient transfers and updates between partners. The model, described by and XML schema, allows the definition of computing systems, including CPU architecture, number of processors, amount of memory and Operating System. Additionally, resource description may be aggregated to reduce the amount of information transferred between sites.

The resource description model is divided between resource elements and their relationships. Listing 4.1 contains an example where a resource of type *ComputerSystem* is associated to another resource of type *OperatingSystem* through a relationship of type *Contains*. This model has enough expressivity to describe complex systems with multiple resources and relationships among them.

As introduced before, however, it is necessary to provide efficient methods for communicating resource information between partners. Our model addresses this need from two different angles. First, the Resource Management API allows partial updates with only information about changes in the resource usage. This reduces the amount of necessary information required when the infrastructure status does not change substantially between updates. Secondly, our transmission model provides compression capabilities by aggregating resource information. Different aggregation models are supported that trade off the amount of information transferred ant the level of accuracy. The most accurate representation model contains all the information from a site, while the highest compression method results in an approximate description of the state. A more in-depth discussion about the supported aggregation methods and their impact in site selection decisions can be found in [BFL+08] and [RGC+09].

4.2.3 Job Broker and Scheduling Criteria

The function of an MS is to optimally forward job submissions to a Local Resource Management System (LRMS) or connected MS in the provider role. A job's requirements for resources and execution environments are expressed by a job submission description document (e.g., JSDL). In our MS design, resource matching for a job is based on three considerations: capacity, capability, and utilization.

The resource models of MSs are described in the schema of Section 4.2.2. The capacity of an MS corresponds to the aggregated capacities of the attached LRMSs plus peer MSs. Capabilities describe what LRMSs and MSs can do: a LRMS capability might include the ability to schedule parallel jobs, or make advance resource reservations. Capabilities at the MS reflect high-level scheduling considerations, domain access, and membership in virtual organizations. MSs in a corporate data center can have attributes stating that it will not accept forwarded jobs during times when important local work must complete. Utilization of resources is considered by an MS when making scheduling decisions. Each MS monitors the utilization of its associated LRMSs, as well as those reported by its peer MSs providing computing power. It is a challenge to measure utilization at the required granularity to make good scheduling decisions. For example, reporting the most recent short-term average CPU utilization of each of thousands of nodes in a cluster may not give the most appropriate information to a scheduler. Less granular methods of utilization reporting are often based on the notion of workload classes. A *class* defines the requirements of a job, such as small, medium, and large; or interactive and batch jobs. For example, a provider can report how many jobs of each class it presently supports, and the occupancy of each class. Additional utilization information can be provided by giving the average waiting time or queue length for each class.

Since we consider aggregated resource information, data involved in the scheduling decision may be less accurate when certain aggregation schemas are applied. Consequently, to perform the matchmaking between job requests and resources from the aggregated data, we take statistical information such as maximum and minimum values contained in the resources for the requirements and a combination of average values for refining the selection. Furthermore, since the resource matching is performed at the broker level, the information loss can result in non-optimal broker selection decisions. For example, the algorithm may select a broker with insufficient resources when another broker is able to dispatch the job immediately. Therefore, the level of aggregation of the resource information is crucial.

4.3 Meta-Scheduler Implementation

The architecture of the MS communication protocol allows for different sites to coexist while maintaining site-specific policies, internal security mechanisms or implementation details hidden from other peers. As long as all MSs implement the required interfaces, the proposed protocol allows them to communicate and share workloads transparently.

As a proof of concept, we have enabled three sites under different administrative domains to communicate using our protocol. Each of these sites has its own implementation in terms of the underlying resource management systems or security and scheduling policies, but all of them implement a common protocol for communication of resource and job information. The site at BSC extends an in-house developed MS, eNANOS, which uses the Globus Toolkit (GT) internally; IBM implements the meta-scheduling protocol on top of one of their commercial products, IBM Tivoli Dynamic Workload Broker; FIU developed an extension to the GridWay open source MS to enable interoperability.

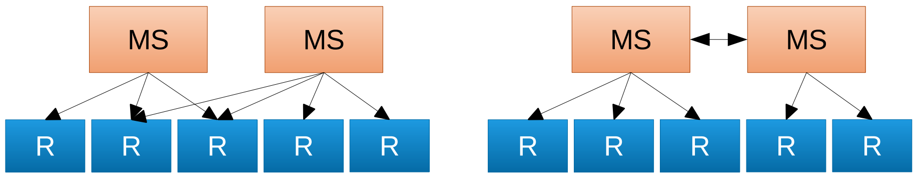


Figure 4.3: Comparison of GridWay and the peer-to-peer model

4.3.1 High Level Architecture

The implementation of FIU's MS leverages an existing open source solution, the GridWay meta-scheduling platform [HML04]. GridWay is a community project with an open source license. We choose it for its use of open standards and its modular nature. GridWay is implemented as a group of managers that communicate using standard I/O. It offers facilities for job management, data transfers and resource information.

GridWay is a Globus incubator project, and therefore, it is supported by the Globus Consortium. This means that all advances in GridWay will be in line with the Globus project. It provides one of the first implementations of the DRMAA (Distributed Resource Management Application) API [DRM] for job submission and status querying.

Originally, GridWay was designed to manage a set of resources that are either controller by an organization, or cross organizational boundaries by using Globus. Multiple VOs may have different instances of GridWay, but the initial implementation does not support collaboration among instances. Contrarily, such instances would compete at the resource level among them. Our extension with the interoperable protocol addresses this shortcoming, as shown in figure 4.3. The left figure depicts the original GridWay model, where instances compete for resources. In the right, we show our collaborative method of resource sharing.

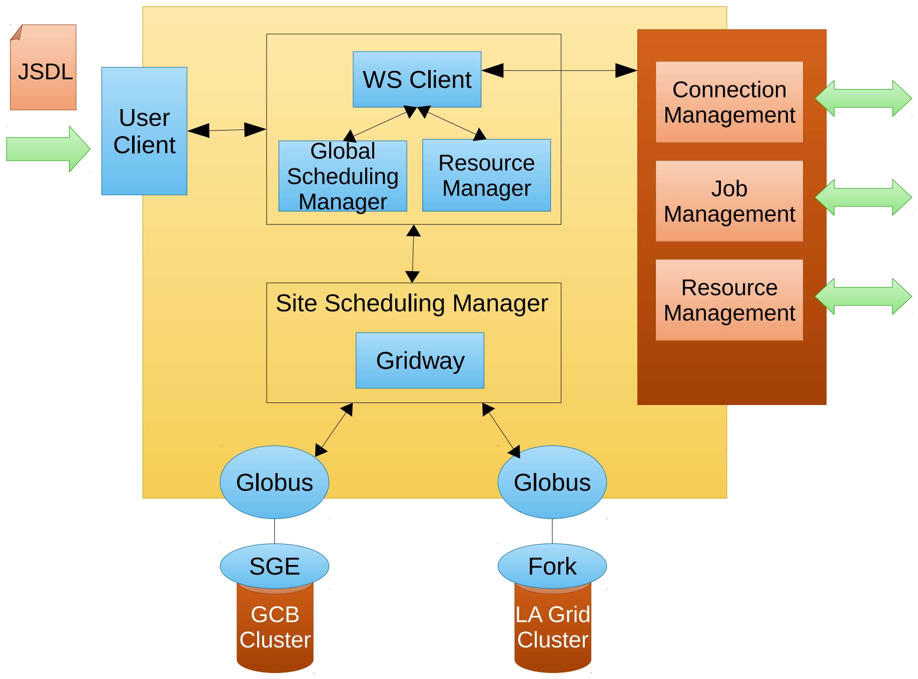


Figure 4.4: Architecture of FIU MS

The FIU MS is built as a set of modules that deal with different aspects such as peer communication, site scheduling, and resource management. The details related to local resource management are delegated to GridWay, while the high-level scheduling decisions are made by our wrapper.

The core modules and their interaction is shown in Figure 4.4. Here is a brief explanation of each module's responsibilities:

- ***User Interface***: The user interface module is in charge of receiving external users' requests such as job submission or resource information. Users can interact with the MS using a command-line interface and Job submission is done through OGF's JSDL files.
- ***Site Scheduling Manager***: This module wraps GridWay functionality for intra-domain scheduling. The rest of the modules interact with it using DR-MAA.

- ***Global Scheduling Manager***: This module deals with global scheduling policies. It is in charge of deciding whether a job will be scheduled and processed on the local domain or it will be forwarded to another domain. Additionally, this module keeps track of the jobs' status submitted from other domains.

- ***Resource Manager***: It stores information about the resources in local domain and the remote domains, which have exchanged their resource information through an open connection. It also pushes resource data to the connected MS in case of changes in local resource availability.

- ***Web Service Communication***: We use Apache Axis2 as the container for the different web services used to communicate with other domains. Axis2 provides the points of entry for SOAP requests and stubs to initiate conversations with other MSs.

4.3.2 FIU Meta-Scheduler Detailed Implementation

In this section we give a more detailed explanation of the MS implementation for the particular case of the FIU prototype. After having described the high level architecture of the system, we will give a more in-depth overview of each of the phases involved in job submission, exchange and execution.

The first phase consists in building a request describing the desired application to execute. For this, users need to write a JSDL document, which can contain elements from the POSIX and SPMD extensions. Listing 4.2, at the end of the chapter, gives an example of such a request document for the execution of WRF. In line 9, the user can assign a name to the job for later identification. The *Executable* element, in line 15, declares the application to be run, and lines 16 and 17 specify where standard output and error will be redirected. Via the SPMD extension elements (lines 18–20), the user can request how many processes to spawn in total, the number of them to run at each host, and the type of parallel middleware, in this case *MPI*. Then, target architecture, operating system, or other host information requirements can be declared through the *Resources* element (lines 23–35). Finally, the input (or *stage-in*) data can be listed with the *Staging* element (lines 36–42 and 43–49). The target MS will fetch those files before executing the application.

Once the JSDL document has been either manually written by the user or automatically generated through a submission client, a SOAP *submitJobFromJSDL* request is sent to the user's local MS. Upon reception of the JSDL document, the FIU MS creates an entry in the job database and returns an identifier to the user, which can be later used to track the job's state.

After the user request has been successfully parsed and accepted by the MS, it is passed to the *Global Scheduling Manager*, which is in charge of making forwarding decisions based on the site's policies. For example, the *LocalFirst* policy, which attempts to execute jobs in locally owned resources first, performs a match between the job requirements and the available resources at the *Resource Manager* to try to run the job locally, or forward it to another site otherwise. In the first case, the control is passed to the *Site Scheduling Manager*, while in the second, the request is encoded to follow the protocol's data specification format and forwarded to another site.

In the case of local execution, the *Site Scheduling Manager* is in charge of translating the request and send it to GridWay. This is performed by creating a *DRMAA Session*, which holds the information encoded in the original JSDL document. GridWay retrieves information from the *Virtual Organization's* owned resources —*i.e.,* all the infrastructure shared by the same middleware with other sites—, selects the target hosts where to run the job using local policies (Round-Robin in our prototype), and submits the job through the *Execution Manager* component, which in the case of FIU is configured to use Globus Toolkit version 4. This middleware defines a Web Service API for the *GRAM* (Globus Resource Allocation Manager) component, in charge of job submission and management.

At FIU, GRAM is configured to use the Sun Grid Engine (SGE [Gen01]) batch system through the SGE-GT4 adaptor [SGE]. The MS is identified through the Globus Security Infrastructure (GSI) by using an X.509 certificate, and Globus translates the JSDL request into an SGE request, analogous to the *qsub* command. SGE supports the submission of MPI jobs through the use of *Parallel Environments*, which orchestrate the execution and lifecicle of MPI applications. If the original JSDL request contained instructions about using a parallel middleware through the SPMD extension, this requirements are translated into the creation of a Parallel Environment. From this point, lower subsystems transfer job state information upwards, until it is received by the MS and communicated to the user or other peering MSs through the Job Management API.

4.4 Meta-Scheduler Validation

In order to evaluate the MS protocol and its different implementations, we first perform two sets of small-scale experiments. In the first set, we measure the timings for different operations of the protocols for three sites with their own implementations. In the second set, we run a scientific software simulating a realistic scenario in the domain of weather forecasting.

FIU	BSC	IBM
FIU meta-scheduler AMD Opteron 2.60 GHz 2 GB RAM	eNANOS meta-scheduler Dual Intel P4 3.60GHz 1 GB RAM	IBM TDWB AMD Opteron 2.60 GHz 2 GB RAM

Table 4.2: Information about the experiment sites

4.4.1 Protocol Measurements

We test the implemented APIs for the IBM, FIU and BSC versions of the MS. The main operations supported by the protocol are tested to perform a functional validation of the MSs. In this experiment, we use a driver program that generates requests for each of the tested operations, and measure the timing. The resources employed at each site are shown in table 4.2. The FIU and IBM MSs are hosted at IBM to overcome a problem with IBM T.J. Watson's firewall, while the eNANOS instance is at BSC in Spain.

For the resource information exchange protocol, each MS aggregates 100 resources and sends them back when it receives the *requestResourceData()* call from the driver program. The driver program then sends the same resource information using the *sendResourceData()* call to MSs. While we have specified the number of resources used for the tests, the type of resources and the aggregation algorithms used vary in different MSs.

For the job execution protocol, we send a probe job defined by a JSDL document that runs the UNIX sleep command for 10 seconds. This allows us to measure the protocol itself, without considering the actual job load.

Operation	Delay Time (milliseconds)			
	FIU → BSC	BSC → FIU	FIU → IBM	IBM → FIU
openConn()	562	659	15	40
requestResourceData()	983	706	69	90
submitJob()	642	694	124	3162

Table 4.3: Delay across meta-scheduling sites

4.4.1.1 Results Discussion

Results show lower latencies between IBM and FIU MS, since they reside in the same network. The FIU implementation has lower turnaround time because connection and job information is stored in memory, while IBM keeps the information in a DB2 database, with the consequent overhead. The FIU MS stores resource information in a file, and for the *requestResourceData()* operation, resource information is read from the file and then aggregated. For the *sendRequestData()* operation, the FIU MS keeps the aggregated data from other MSs in memory without file operations. For *submitJob()*, the FIU MS routes the job to GridWay and obtains a job ID before returning to the caller.

Compared to two other MSs, BSC has longer delays in most of the operations. The critical factor is the additional delay produced by the WS wrapper between other MSs and the eNANOS LA Grid service.

For the *requestResourceData()* operation, the delay time includes having resource information retrieved from the *Resource Properties* without actual resource discovery. There is a background eNANOS service responsible for resource discovery and populating the resource properties at a configurable interval. The retrieved resource information is then transformed into an aggregated form and packaged as part of the returned SOAP message. The *sendResourceData()* operation involves depositing the resource information in the aggregated form to the Resource Properties.

The IBM column in Table 4.3 shows the data collected on interactions between the driver program and IBM MS. The *submitJob()*, *requestResourceData()* and *sendResourceData()* operations involve access to multiple tables in the database such as for storing job information, and retrieving and storing resource data. The data shows that the *sendResourceData()* operation consistently takes longer than *requestResourceData()*. Delays are probably caused by the database update overhead, as resource data received by the *sendResourceData()* operation is also stored.

4.4.2 Weather Research Scenario

Interoperability among the three MS implementations is verified by running the Weather Research and Forecasting (WRF) model [MDG+04]. WRF is developed by the National Center for Atmospheric Research and several other research institutes as a tool for meteorologists to do regional forecasting. The WRF model is an MPI application that follows the SPMD (Single Program, Multiple Data) paradigm. Blind scaling of WRF across the grid is generally counter-productive and in most cases degrades the total running time of the model [MWZS09]. This is due to a design which assumes a low latency underlying network and uses intensive communication among working processes.

Different approaches to scaling out WRF have been discussed in [SFB+08], of which meta-scheduling is a paramount component in the process. On this experiment, we show the advantages of running jobs through the MS compared to running them directly on a local cluster. The MS can extend available resources by connecting to other peers and delegating jobs to them based on different policies. Users do not need to do any additional work, since the protocol takes care of contacting other sites, keeping track of remote resources and forwarding the jobs.

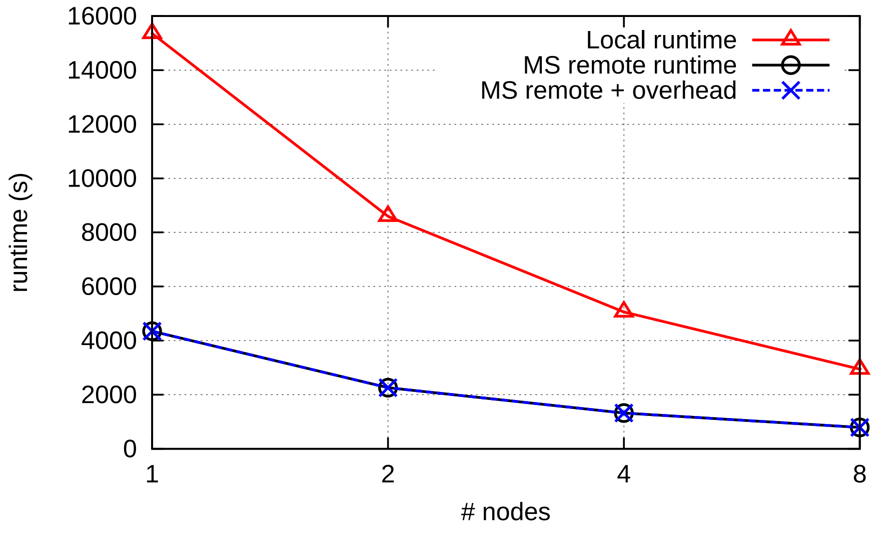


Figure 4.5: WRF execution results

We compare three cases to assess the run-time improvements of delegating jobs, and measure the overhead of the MS layer.We performed 4 different runs of a small forecasting region for 1, 2, 4 and 8 processors.

All experiments include two execution sites with different resources. The local site consists of 8 Pentium 4 nodes at 3 GHz with 1 Gb of RAM located at Florida International University in Miami. The remote cluster has 4 IBM JS22 blades with two 4GHz dual core Power 6 processors and 16 Gb of memory, and is located at IBM TJ Watson center in New York.

In the first case, we measure the running time of the different WRF instances at the local site without using an MS. The user executes WRF through *mpirun* to spawn parallel tasks in the cluster.

In the second case, the same jobs are executed, but this time through an MS installed at the local site. Instead of running the job through logging in the node and issuing the mpirun command, now the user constructs a JSDL definition file with the command, and specifies how many nodes he/she wants to allocate for the job. We use the SPMD extension of JSDL [ABD+05] to describe the parallel environment to be used (MPI in this case) and the number of nodes to use. The MS is configured to use the local cluster to run the job, obtaining similar results as in the first case.

Finally, the four WRF instances are sent to the same MS at the local site, but in this case we change the scheduling policy to forward jobs to another peer. The user submits the job using the same method of submission and job description file as in the previous case, but this time the MS contacts the remote instance running at IBM and forwards the jobs there. We assume that the job's binaries are located in both sites, so data transfer is not considered here.

Figure 4.5 shows the run-times for the execution of WRF at the local site and the execution of the same experiments through the MS with jobs forwarded to the instance at IBM. As it can be noticed, the processing and network overheads when using the forwarding policy –an average of 1.74 seconds– are negligible for a scientific job such as this one. The experiments show how it is possible to scale computing resources by transparently delegating execution to external sites with better equipment.

4.5 Large-Scale Meta-Scheduler Evaluation

A challenge to properly evaluate the effectiveness of MSs is the complexity of developing a realistic grid experimental environment. In this section, this challenge is addressed by a unique combination of two approaches: first, we develop a LRMS emulator to provide a flexible and scalable model for local resources of a grid environment. Second, we use reduced workload traces to demonstrate the resource matching and scheduling functions of the MS. Real workload traces are reduced while preserving their key workload characteristics to allow exploration of various dimensions of MS functions in reasonable time.

Given the large size and heterogeneous nature of grid environments, different strategies have been devised towards its study. Examples of these strategies are small deployments in controlled environments, or the use of analytic or simulator models. However, as noted by researchers such as Iosup [II$^+$05], the size of production grids makes realistic analytic modeling hardly tractable. Evaluated characteristics such as user requests, computing resources and different implementations present significant challenges for simulators to capture the full behavior of all involved components. Also, the nondeterministic and other dynamic behaviors of grid make simulation approach less suitable.

In this section, we propose an approach that consists in real MS implementations with emulation of grid physical resources. We select emulation rather than simulation as typical event simulators don't allow simulated resources to communicate with the existing code. Additionally, emulation reproduces the behavior of a component in the system so that externally it appears to behave as the component itself. We can replace an emulated component with a real implementation without influencing other components in our grid system.

Our LRMS emulator provides job scheduling on clusters of compute nodes whose scale and power can be defined as parameters. This enables experiments with multiple grid configurations by "plugging" LRMS into the MSs domain of control without modifying the MS implementation. The MS sees the virtual resources provided by the emulator as real ones. This greatly enhances the variety of tests that can be performed under different cluster configurations. Most importantly, by retaining the real MSs on their actual physical networks, it is possible to preserve the large scale structure of the grid that we use to study various interaction hahaviors.

The concept of using emulated resources and their allocations was also previously mentioned in [JHFS01] for cluster scheduler evaluation. We extend the

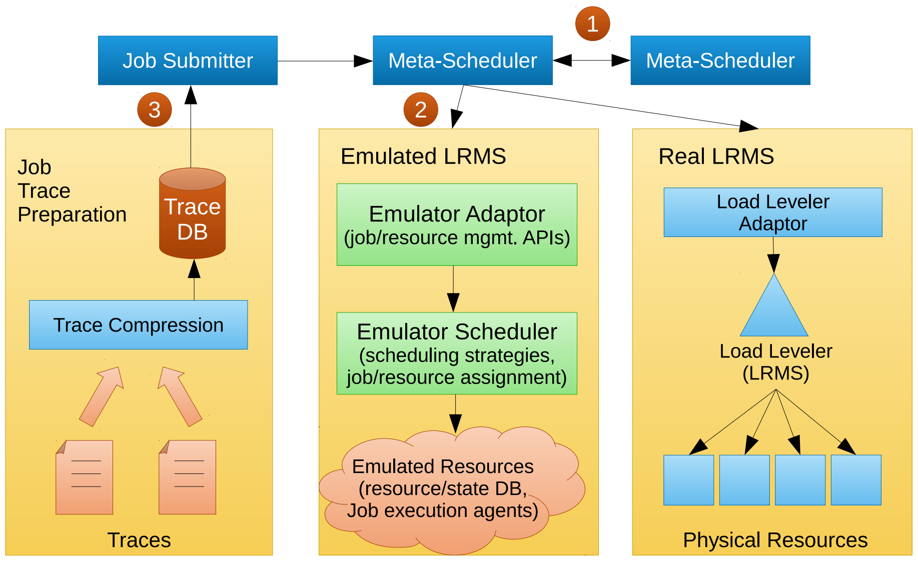


Figure 4.6: Meta-Scheduler evaluation architecture

emulation to multiple grid clusters for the MS evaluation of varying characteristics. Several other research groups developed grid simulation frameworks and published several papers evaluating the various aspects of grid scheduling algorithms [ITF+08b, ATN+00, RGC+08b, BDF+08]. Performance studies using real systems offer additional insights that are hard to learn from analytical or simulation methods.

4.5.1 Meta-Scheduler Evaluation Platform

Figure 4.6 shows an architectural overview of the prototype MS evaluation platform used in this section. Instances of MSs are deployed at multiple resource domains. Each MS implements the interoperability protocols described previously in Section 4.2. Based on brokering policies and resource information exchanged amongst peer brokers, jobs entered to a domain are executed locally or routed to remote MSs. Each MS interacts with one or more LRMS to allocate resources for the jobs. As shown in Figure 4.6, LRMSs can be either real systems (e.g., IBM LoadLeveler [Loa]) managing real physical resources or emulated. Job traces are used to generate job workloads by a job submitter from any client site. The functional steps of the experiment are indicated in the figure by the labels: (1) Job forwarding to another MS, (2) Job submission to a LRMS, and (3) Job submission into the system by a user.

4.5.1.1 Emulated Local Resource Management

As shown in Figure 4.6, the LRMS emulator component consists of three main parts: the Emulator Adaptor, the Emulator Scheduler, and the Emulated Resources. Note the equivalency of the functions in the emulated LRMS to those of the real LRMS (IBM LoadLeveler or any other generic LRMS). The emulation adaptor interacts externally with the MS for job management (e.g., submission, termination, and query) and resource information exchange. For example, the resource information may include types and numbers of resources available in LRMS, and various utilization information (e.g. statistics on job queue lengths and system utilization).

The second part of the emulated LRMS implements the behavior of a local scheduler (e.g., batch scheduler like Condor, PBS, SGE, or LoadLeveler) that performs assignments of jobs to resources managed by the LRMS. Our current implementation of the emulator scheduler is the First-in-First-Fit algorithm. In this scheduling discipline the job queue is always sorted by arrival time. However, when the scheduler processes the queue, if there are insufficient resources for the first job to run, the next job is considered and will run if sufficient resources are available, and so on until a job can be run or the queue is exhausted. This scheduling policy keeps the resources utilized, but has the potential starvation of large jobs. More complex algorithms like backfill scheduling can mitigate the potential of resource starvation for jobs requiring large number of nodes.

The third part of the emulated LRMS provides virtual resources from a configuration file, as in Listing 4.3.

It keeps the current state of each resource in terms of OS types, processor architecture, memory, and disk utilization. We assume that each job utilizes completely a processor, and that different jobs don't share the same set of processors. This may seem unrealistic in terms of emulating the performance of an application in a given machine (the impact of the instruction set, memory access speed, and I/O interrupts). However, the goal of the emulator is not to give an accurate representation of how fast an application runs on a given set of machines, but to understand the characteristics of resource utilization and job services.

4.5.1.2 Workload Trace Reduction

We have chosen to drive our experiments from real job traces representing a scientific workload. The trace is selected from the Parallel Workloads Archive [Fei08] and contains 11 months of execution at the Cornell Theory Center (CTC) on a cluster of about 450 single processor IBM SP2 nodes with similar CPU, memory and disk. However, given the size of this trace, it would be prohibitively expensive to run it in real time. Therefore, we perform a technique called trace reduction on the workload, to bring it to manageable size while keeping its key characteristics.

Trace reduction techniques have been used by computer architecture designers to shorten time for simulations for microprocessor design. Authors in [EGB03] compared the approaches of sampling and reduced input sets by using different techniques, such as reduction in repetitiveness and input truncation, while maintaining statistical similarity to the original input traces

Real-time execution is used to allow the emulated resources to interact with the physical system components. In order to conduct experiments based on long traces in reasonable time the traces need to be compressed in the time domain. A heuristic approach in reduction is taken and shown to retain several relevant properties of the original trace. Because the timescale of MS scheduling is long (many seconds) it is unnecessary to preserve the fine scale details of the original trace. The initial work reported here targets an execution time of a few hours and is derived from samples of about one weeks data from the original traces. The process has three stages:

1. **Select a sample interval from the original trace**. The archive traces contain many months of data. So the first step is to select an interval representative of the entire trace. This interval is compressed in subsequent steps. The CTC trace, which corresponds to a cluster of homogeneous computers, contains well defined and cyclical periods of submissions with duration of a week. Therefore we select a period of activity in which this behavior can be observed. We avoid using intervals that belong to the warm-up or wrap-up phases of the system.
2. **Trace sampling**. After the trace interval is selected, a subset of jobs is chosen from this interval. This subset size is not fixed but depends on the sampling rate, which considers the target resource size appropriate in the emulated cluster experiments. Because the CTC trace is from a cluster at Cornell of about 450 machines, we decided to use this same size of resources in our test environment. We further reduce the number of jobs such that the total execution time is a few hours by job sampling from the weekly trace, for example taking only 2 out 3 jobs. The requirements of each remaining job are preserved in this phase. In particular, the number of requested processors, the job submission time, and the job execution time are left unchanged.

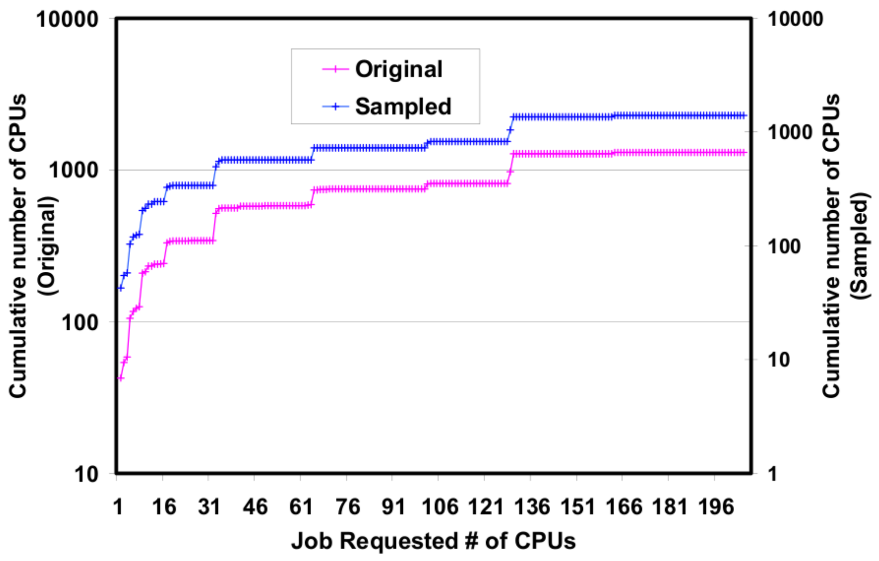


Figure 4.7: CTC original and reduced traces

3. **Time scaling**. In this phase independent scaling factors are applied to the job inter-arrival and execution times. Although for the experiments reported here a common scale of 60 is applied to both.

Finally, the properties of the traces prior and subsequent to reduction are compared. Figure 4.7 shows the cumulative distribution of processors, which indicates that the ratio of requested CPUs stays the same after modifying the trace. Then we perform a hierarchical clustering analysis with average linkage to find the submission trends of the trace by grouping jobs that were submitted in similar time periods. The distance between clusters is defined by using the difference between submission times. We heuristically determine k, the number of clusters, by plotting the total within-cluster sum of squares (WCSS) for different values of k and then finding the value for which the WCSS has a smaller increase, or an "elbow", in the graph. This and other methods are discussed in [Har75]. Figure 4.8 shows that clusters are closely correlated between the original and sampled traces: this indicates that sampling retains the arrival time distribution of the original workload.

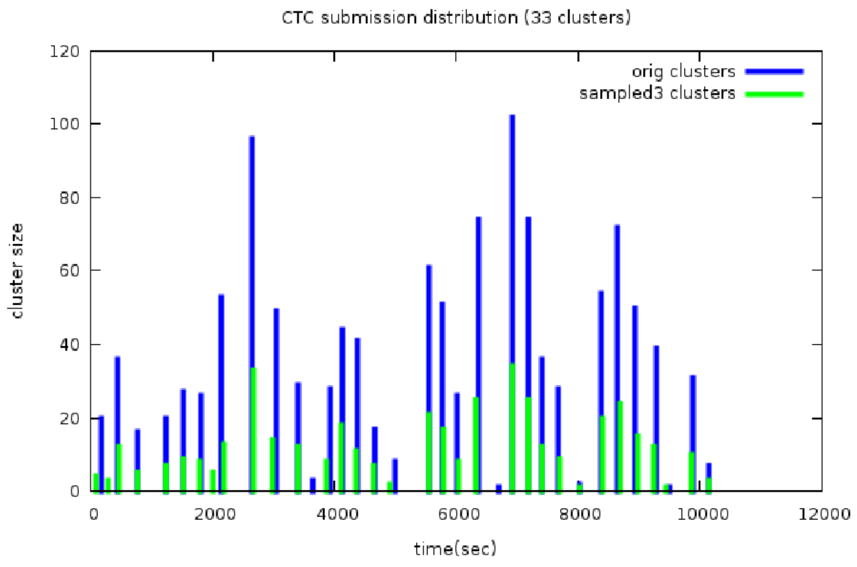


Figure 4.8: Cluster analysis of original and reduced traces

4.5.2 Experimental Results

The goal of the experiments in this section is to quantify the behavior of the distributed MS model, compared with centralized approaches where all resources are under one single domain. The advantages brought by our solution have already been described: site independence and interoperability, flexibility in sharing and scalability over multiple domains. However, there is an overhead due to the protocol's distributed nature. Peers need to exchange their resource information, and resources are divided among collaborating sites.

In the current experiments, we use the described emulation and trace reduction methods to compare the execution of the CTC trace over multiple sites and only using a LRMS, implemented by our emulator. The additional MS overhead is measured to characterize its behavior.

The first experiment runs the CTC trace on one domain with as many resources as the original site, this is, 450 single-processor IBM SP2 machines. Then, we

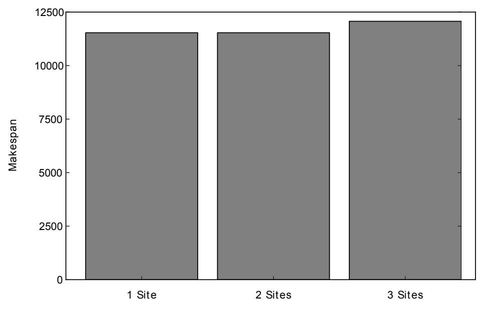


Figure 4.9: CTC Makespan for different number of sites

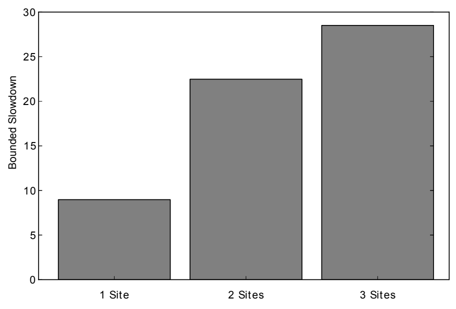


Figure 4.10: CTC Avg. Bounded Slowdown for different number of sites

perform the same run dividing the machines over two sites, with 225 machines each, and finally over three sites with 150 machines each. For this experiment, we can't reduce the number of machines further, since the maximum number or requested processors is 128. Each site uses the First-In-First-Fit allocation policy, and MSs forward jobs among them using the RoundRobin policy where there is more than one site.

Figures 4.9 and 4.10 show the results of this experiment. As it can be appreciated, the makespan of the trace remains almost the same for the different configurations, therefore, we can conclude that the execution's duration for this trace is not affected by distributing the resources among sites. However, a closer look to the results shows a higher slowdown when MSs are used. This is produced by the fragmentation of resources, and results in higher queue waiting times at different sites.

Consequently, it is clear that better forwarding strategies are required to make a better usage of distributed resources. For the next experiment, we explore forwarding strategies that try to improve site-selection. First, we implement `LowestQlen`, which sends a job to the site with the smaller number of jobs awaiting for execution. The second policy, `LowestQlenCpus`, considers not only the number of jobs, but the

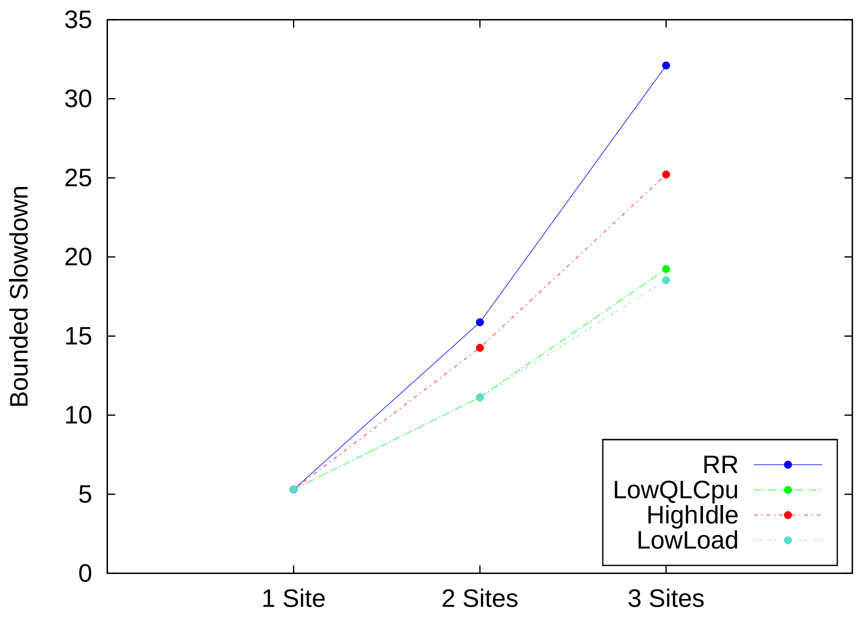


Figure 4.11: Bounded Slowdown for compared forwarding policies

sum of their requested processors. `HighestIdle` chooses the site with higher number of unused machines, and `LowestLoad` selects the site with the lowest load. We define load here as the aggregated number of CPUs in the queue *minus* the number of available resources.

Figure 4.11 shows that policies that account for site usage give much better slowdown metrics, while maintaining the same makespan for the workload. In particular, we can conclude that policies that take the total number of CPUs in the queue give the best results, approximately two times higher than having all resources under the same control in the worst case. This means that the price to pay for using distributed resources through a MS in this case is that on average, jobs have to wait for double time when exactly the same number of resources are considered. This, however, is an edge case, since usually, when adding new sites to the grid, available resources tend to increase rather than stay the same.

4.6 Conclusions and Future Work

This chapter introduces a cooperating MS model that has been implemented by three partnering institutions: Barcelona Supercomputing Center's prototype using eNANOS, IBM Research prototype using the IBM product ITDWB, and Florida International University's prototype using the GridWay from the open source community. Our current work focuses on the MS model and the mechanisms to support cooperation: a set of protocols to connect the MSs, submitting jobs between them, and resource information exchange. The data collected from the different implementations is intended to validate the operations between the three sites.

Furthermore, we define a novel method to test the distributed MS architecture, the instances' implementation, and the choice of forwarding policies. We describe how emulation of computing resources plus trace reduction techniques can be used to compare forwarding strategies, and help researchers choose better policies to decrase job waiting time and execution makespan.

The presented prototype serves as a platform for our current research activities in the area of meta-scheduling, and will allow the exploration of new functions and protocols to optimize job matching to remote domain resources. Our platform has been used by LA Grid partners to explore applicability of grid computing in areas such as hurricane prediction, bioinformatics, and healthcare [B$^+$07].

```
<Resources>
  <Resource>
        <ResourceID>
          <Type>ComputerSystem</Type>
          <Name>CS_0</Name>
        </ResourceID>
        <Attribute>
          <Name>NumOfProcessors</Name>
          <Value>2</Value>
        </Attribute>
        <Attribute>
          <Name>ProcessorType</Name>
          <Value>x86</Value>
        </Attribute>
  </Resource>

  <Resource>
        <ResourceID>
          <Type>OperatingSystem</Type>
          <Name>OS_0</Name>
        </ResourceID>
        <Attribute>
          <Name>OperatingSystemType</Name>
          <Value>LINUX</Value>
        </Attribute>
        <Attribute>
          <Name>TotalPhysicalMemory</Name>
          <Value>1024.0</Value>
        </Attribute>
  </Resource>
  <Relationship>
        <Type>Contains</Type>
        <SourceResourceID>
          <Type>ComputerSystem</Type>
          <Name>CS_0</Name>
        </SourceResourceID>
        <DestinationResourceID>
          <Type>OperatingSystem</Type>
          <Name>OS_0</Name>
        </DestinationResourceID>
  </Relationship>

</Resources>
```

Listing 4.1: Resource description example

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition xmlns="http://edu.fiu.cs.ms"
                    xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
                    xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/
                        jsdl-posix"
                    xmlns:jsdl-spmd="http://schemas.ogf/jsdl/2007/02/jsdl-
                        spmd">

  <jsdl:JobDescription>
    <jsdl:JobIdentification>
      <jsdl:JobName>WRF</jsdl:JobName>
      <jsdl:JobDescription>WRF sample run</jsdl:JobDescription>
    </jsdl:JobIdentification>
    <jsdl:Application>
      <jsdl:ApplicationName>wrf.exe</jsdl:ApplicationName>
      <jsdl-spmd:SPMDApplication>
        <jsdl-posix:Executable>/home/wrf/wrf.exe</jsdl-posix:Executable>
        <jsdl-posix:Output>wrf.out</jsdl-posix:Output>
        <jsdl-posix:Error>wrf.err</jsdl-posix:Error>
        <jsdl-spmd:NumberOfProcesses>4</jsdl-spmd:NumberOfProcesses>
        <jsdl-spmd:ProcessesPerHost>2</jsdl-spmd:ProcessesPerHost>
        <jsdl-spmd:SPMDVariation>MPI</jsdl-spmd:SPMDVariation>
      </jsdl-spmd:SPMDApplication>
    </jsdl:Application>
    <jsdl:Resources>
      <jsdl:CPUArchitecture>
        <jsdl:CPUArchitectureName>PPC</jsdl:CPUArchitectureName>
      </jsdl:CPUArchitecture>
      <jsdl:OperatingSystem>
        <jsdl:OperatingSystemType>
          <jsdl:OperatingSystemName>LINUX</jsdl:OperatingSystemName>
        </jsdl:OperatingSystemType>
      </jsdl:OperatingSystem>
      <jsdl:IndividualPhysicalMemory>
        <jsdl:LowerBoundedRange>2048</jsdl:LowerBoundedRange>
      </jsdl:IndividualPhysicalMemory>
    </jsdl:Resources>
    <jsdl:DataStaging>
      <jsdl:FileName>wrfbdy_d01</jsdl:FileName>
      <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
      <jsdl:Source>
        <jsdl:URI>http://skywarp.cs.fiu.edu/wrf/wrfbdy_d01</jsdl:URI>
      </jsdl:Source>
    </jsdl:DataStaging>
    <jsdl:DataStaging>
      <jsdl:FileName>wrfinput_d01</jsdl:FileName>
      <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
      <jsdl:Source>
        <jsdl:URI>http://skywarp.cs.fiu.edu/wrf/wrfinput_d01</jsdl:URI>
      </jsdl:Source>
    </jsdl:DataStaging>
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

Listing 4.2: Example JSDL request

```
<EmulatedResources xmlns="http://cs.fiu.edu/emulator/resources">
  <Resource>
    <Count>64</Count>
    <Architecture>x86</Architecture>
    <CPUCount>2</CPUCount>
    <OS>LINUX</OS>
    <PhysicalMemory>1024</PhysicalMemory>
  </Resource>

  <Resource>
    <Count>32</Count>
    <Architecture>powerpc</Architecture>
    <CPUCount>4</CPUCount>
    <OS>AIX</OS>
    <PhysicalMemory>4096</PhysicalMemory>
  </Resource>
</EmulatedResources>
```

Listing 4.3: Resource emulator configuration

CHAPTER 5

A MULTI-LAYER FEDERATION ARCHITECTURE FOR THE CLOUD

In this chapter we take a more general view of the problem of sharing workloads across partnering organizations. We explore how software, platform and infrastructure providers can collaborate by defining a multi-layered federation architecture where communication takes place at each service layer through a broker component. We explore the delegation of requests between layers in an organization and among partners at the same layer, or federation. Then, the model is motivated by studying the delivery of a scientific application, the Weather Research and Forecasting service (WRF), across the software, platform and infrastructure layers. WRF is used to illustrate the concepts of delegation and federation, the translation of service requirements between service layers, and inter-cloud broker functions needed to achieve federation.

5.1 Introduction

With the aid of cloud computing technology, businesses and institutions make compute resources available to customers and partners to create more capable, scalable, flexible, and cost effective environments for application development and hosting. Cloud computing continues the trend started with on-demand, strategic outsourcing, and grid computing, to provide IT resources as a standardized commodity, targeting real-time delivery of infrastructure and platform services. A next step in this evolution is to have cooperating providers of cloud services in which a customer request submitted to one cloud provider is fulfilled by another, under mediation of a brokering structure (e.g., [RB08]). This latter idea invokes a federation of cloud domains providing a service analogous to that of interoperating grid resources created for a similar goal by research institutions using grid brokers in the grid computing framework.

Figure 5.1 is an example of what is meant by a federated cloud structure mediated by brokers. The figure shows two independent clouds, each supporting a vertical stack of service layer offerings from the software or application layer (SaaS or AaaS) at the top, through the middleware or platform layer (PaaS), to the operating system and infrastructure layer (IaaS). At each layer a choice is made to fulfill a service request through local resources using *delegation*, or by a partner cloud through *federation*. A key feature of our model, is that federation occurs between cloud providers at matching layers of the service stack.

To illustrate how this works, consider a business providing a SaaS offering from a private or public cloud. Users submit requests to the application layer which assesses if sufficient local resources are available to service the requests within a specified time. If the application layer cannot meet its service goals it can optionally fulfill the requests through an independent SaaS layer provider of the same service as indicated by the horizontal (*federation*) line connecting cloud A to B. Results are returned to the user as if locally produced by the application executing in cloud A. Federation at the SaaS layer is analogous to the use in traditional business of 'sub' or 'peer' contractors who supply equivalent final parts or services to the primary provider facilitating elasticity to support a dynamic market. While this approach is common in industry sectors that produce goods or services such as manufacturing or publishing, it is not as common in software due to lack of standard interfaces and insufficient market forces to motivate sharing at the service layer.

An application layer under stress also has a second option to increase capacity through *delegation*. In this service abstraction, the application layer works together with its underlying layers to provide the required computing needs. In delegation,

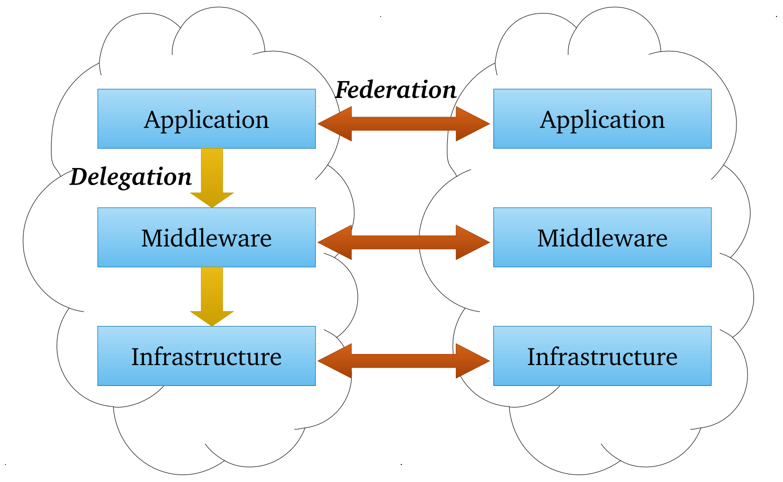


Figure 5.1: Federation and delegation in cloud application support

the application layer asks the PaaS layer in the local cloud for additional resources. The request for more resources may be fulfilled in multiple ways depending on availability in the current cloud. The PaaS layer can delegate to the local IaaS layer a request for more raw virtual machines and then provision the necessary platform software. If sufficient resources are not available locally the PaaS layer can attempt to acquire them from another cloud in the federation through brokering at the PaaS layer.

In a typical scenario, the PaaS layer represents executing middleware such as web application containers and other application execution platforms, or distributed data applications. Here a more general view of federation is needed in which these support programs and environments form the federations between the clouds in a way that isolates them from the underlying infrastructure layer. Some current middleware products, such as web application servers (e.g., IBM WebSphere Application Server or Oracle Fusion Middleware), provide isolation or lightweight virtualization from the underlying hardware and allow applications to dynamically expand across machines increasing capacity.

While attractive from a business perspective, this federated cloud model requires new technologies to work efficiently. Because it is a layered model, an important part of the design is to maintain isolation of concerns between layers. For example, the SaaS application delivers a result to the customer in a certain response time. It is aware of the aggregate processing and network transmissions necessary to meet the delivery time. But the application does not need to know the details of the underlying infrastructure. Thus, it is necessary to translate requirements at the application to those understood by the PaaS and IaaS layers. This is accomplished through empirical modeling and experiments that map metrics of application performance such as response time onto the middleware and compute resource requirements understood by the PaaS or IaaS layer.

One challenge to making the operation of delegation work is to introduce a standardized form of expressing inter-layer mappings. Some work along this line is contained in the *manifest* approach used by the Reservoir project [RBL+09]. In Section 5.5, we discuss some parameters that need to be translated across layers, in the context of the application we use as a case study. A related issue is how to choose between delegation and federation when both options are available. Selection criteria such as the mapping of performance metrics may be combined with policies as discussed in Sections 5.2 and 5.5. Another challenge is defining the protocols and policies for the inter-cloud brokering required to join each layer in a federation. Section 5.4 considers brokering at different cloud service layers and then proceeds to the inner workings and policy issues by which brokers expose and share cloud services and resources.

It is difficult to fully understand the federation model of Figure 5.1 without a concrete example. Because of our experience with parallel and distributed computing, we choose for this purpose the Weather Research and Forecasting (WRF) software as a service (SaaS). WRF is a batch mode service in which customers request weather forecasts over a region with a specified level of detail/resolution. It provides a good case study for cloud hosting as it is a high performance computing application for which the private and government agencies that use it would like to leverage their joint resources through cloud services. Section 5.5 is devoted to how to implement the model of Figure 5.1 within the context of providing WRF as a service. This study offers the opportunity for interesting contrasts with the previous chapter, which considered federation of grid infrastructure. In those experiments, multiple HPC sites can accept a WRF job submission and a distributed system of peer brokers routes customer WRF requests input to any of the sites to the one providing the best response. Here, a single site hosts the WRF interface to the customer at the SaaS layer, and additional PaaS or IaaS resources are brought under the control of that site when needed to meet performance requirements.

5.2 Cloud Service Stack Architecture

There is an extensive list of works in the literature classifying the services offered by cloud providers in various ways. A common feature of these cloud models is the layered service model where each layer provides an increasing abstraction and isolation from its underlying layer, progressing from raw hardware to software and ending at the application layer. For example, [TMC$^+$08] shows the cloud architecture as layers of Hardware as a Service (HaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a service (SaaS). [LKN$^+$09] categorizes the cloud as a stack of service types, namely IaaS, PaaS, SaaS, Human as a Service (HuaaS), and Support Services. The Reservoir architecture [RBL$^+$09] uses three layers, the Virtual Execution Environment Host (VEEH), virtual Execution Environment Manager (VEEM), and Service Manager. While the names are different, there is a close functional correspondence of the Reservoir layers to IaaS, PaaS and SaaS.

The reference cloud model presented here adopts the three service layer model of Figure 5.1. This model defines layers according to clearly specified principles such as isolation, abstraction, elasticity, runtime and fault tolerance. More details will be presented in Section 5.2.1.

One significant idea of this chapter is that inter-cloud federation is constrained to occur only at corresponding layers of the reference model. Section 5.2.3 argues for this point of view and leads to the discussion of how federations are created and brokered in Section 5.4. The layered federation model contrasts with the Aneka Federation [RB08] and Reservoir Federated cloud [RBL$^+$09]. For Aneka Federation, each cloud site instantiates a service component called Aneka Coordinator which basically implements the resource discovery and management functions. It is our view that the Aneka Coordinator is responsible for the federation functions at the IaaS layer. For the Reservoir model, the federation function is supported through VEEM-to-VEEM communication, thus supported at the PaaS layer only. Delegation involves a request translation mechanism in order to convert the requirements from one layer to another. Translation is a complex and layer dependent function explored more in 5.2.2.

There are several implications in our model when comparing it to existing work or other possible approaches to resource sharing. It can be argued that layered, peer-to-peer federation adds additional complexity to the negotiation and execution of tasks among different providers, since each site needs to implement different protocols and translation mechanisms. However, we believe that the added flexibility justifies this additional work. First, by defining different federation methods at each layer, we allow elastic and fault-tolerant behavior at different stages of the process.

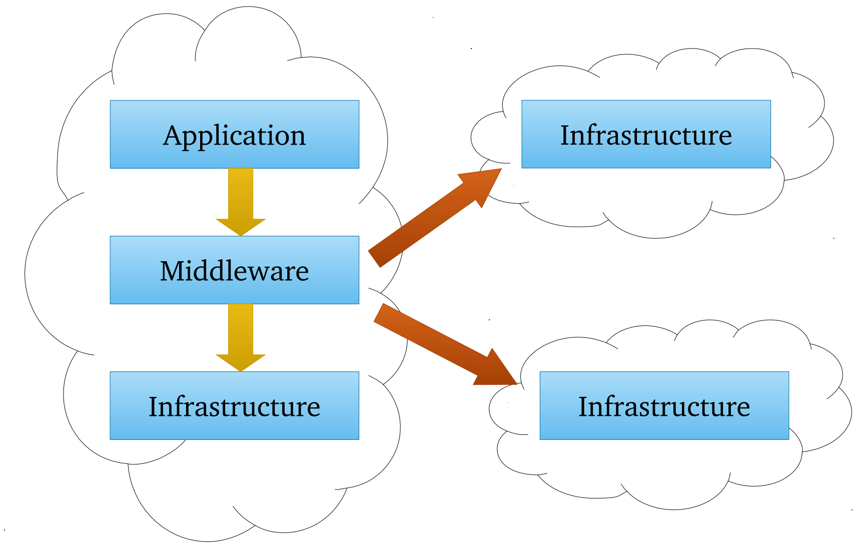


Figure 5.2: An alternative model of federation

Second, providers can focus on concrete aspects without having to implement the full layer stack (for example, we could imagine specialized providers that only offer functionality at one layer). This aspect expands the possible interaction with other systems, not restraining them to a given model. Finally, the decentralized, distributed federation model is very applicable to the current cloud model, where new providers rapidly appear and need to easily join other working systems.

As an alternative, a federation architecture would allow a layer to offload work to underlying layers either in the local or in different providers, as shown in Figure 5.2. In this example, the Platform needs to handle two variations of delegation protocols —both on and off site—, instead of one delegation and one federation protocols. The shortcoming of this model is the lack of elasticity of platform resources through federation of platforms. In contrast, the federation of analogous layers eliminates the need of different types of delegation and enables elastic capacity at the platform and infrastructure layers.

We further sharpen the differentiation of layers by studying how they function to support cloud applications. In particular, how layers work together within a cloud to support an application, while inter operating with peer clouds to provide additional elasticity to application capacity.

Figure 5.3 demonstrates the WRF application example. In the figure we can see three different layers at a given provider that implement distinct capabilities. Each of the layers is in charge of managing its corresponding input data, assessing whether the request can be processed locally through delegation or if it should be sent to another site through federation. In the first case, the request needs to be translated so that it matches the expected input of the next layer in the stack. In the second case, a brokering module at the federated layer needs to establish a connection to another provider and negotiate the terms for which the tasks will be accomplished. Choosing the optimal run time option is a non-trivial problem requiring that requires taking into account the cost and computational requirements of the desired service. In Section 5.5, we discuss our vision for addressing this problem using application performance modeling.

5.2.1 A Layered Model of Cloud Services

The top layer is the software layer, which deals with requirements in executing the application within the context of the key performance metrics (KPM) of the application offering in addition to application execution environment. For WRF this exemplary KPM is completion time for a weather forecast of a user specified geographic region with a certain resolution. The application service layer is aware of the KPMs and software and how they translate into resources at the PaaS. The information for this mapping from KPM at SaaS to PaaS resources is developed through off line experiments and input from online results.

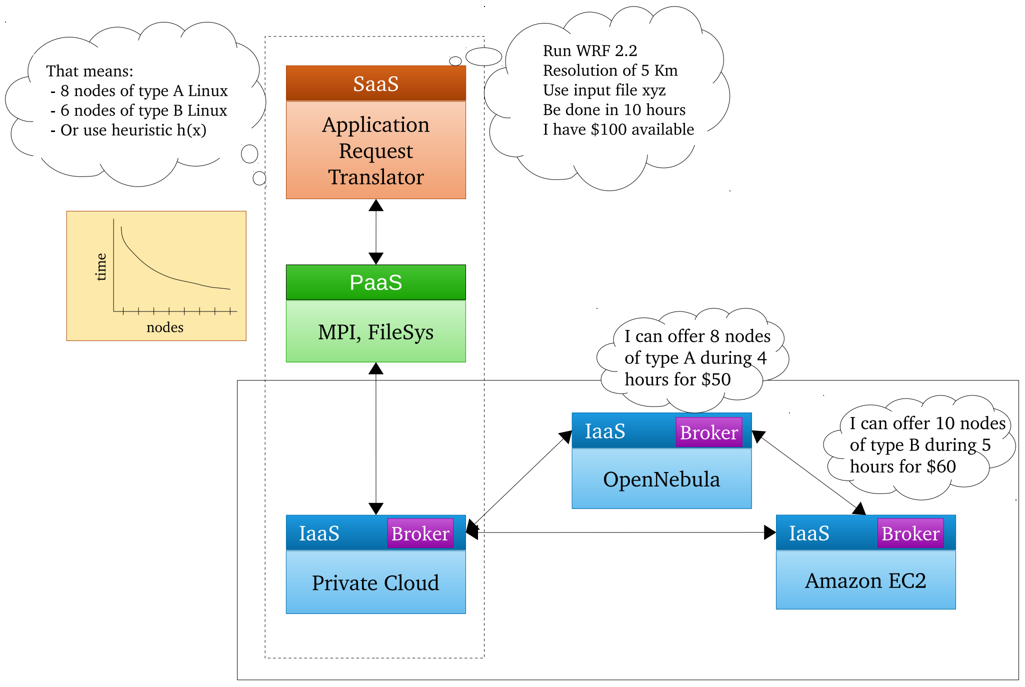


Figure 5.3: Conceptual model

The next layer in the stack corresponds to the Platform as a Service layer. This is traditionally the most overloaded term in the cloud. Specifically, we define the intrinsic characteristics of a PaaS provider:

- **Development library** A PaaS offering allows a developer to build the target application by using a defined library
- **Runtime environment** The platform has a runtime component that manages the application's underlying aspects
- **Layer decoupling** It is decoupled from the upper and lower layers. This means that, first, the platform layer does not have any knowledge of the application specific details. Second, it is agnostic to the underlying infrastructure
- **Elasticity and Fault tolerance** Finally, the platform layer needs to support operations that will result in the cloud's elastic behavior. This means that it

needs to allow scalable resource allocation and have mechanisms to deal with failures

The PaaS layer corresponds to the traditional concept of middleware and represents the bridge between application requirements and elastic infrastructure resource management. This layer does not consider the actual infrastructure —*e.g.*, how many Virtual Machines need to be provisioned—, but rather a higher representation of execution units such as tasks, processes, threads, *etc.*.

Well known examples of PaaS offerings in the cloud are Google App Engine and Microsoft Azure. However, this layer can be implemented by different means. An example of this in the WRF application stack would be MPI [GHLL+98]. MPI is both a development library and a runtime environment, it does not consider either application specific details nor make assumptions about the underlying resources, can be executed for a varying number of processes, and offers a simple fault tolerant behavior (by terminating a job when one of the processes fails). The newer specification of MPI-2 [GGHL+96] includes further features to dynamically add and remove MPI tasks to/from running applications and thus would be useful in exploiting the elasticity capability of cloud resources.

Finally, the IaaS layer represents the resources of infrastructures on top of which the rest of the stack is supported. The concepts managed at the IaaS layer correspond to Virtual Machines, disk images, network connectivity and number of processors, for example.

5.2.2 Inter-Layer Delegation

Cloud provider sites can support different layers of functionality, and not all uses of the cloud need to traverse all possible layers. However, if we consider an application hosted in the cloud, it is useful to study all stages involved in the process, since they will have an impact on different aspects such as price, performance, fault tolerance, etc. The lifecycle of a cloud application includes all layers, either implicit or explicitly. Policies are transferred from one layer to the next one, and failing to fulfill them on a single layer is likely to affect the capability of other layers to offer the required service.

The user of such an application initiates the interaction at the Software as a Service layer. Requirements at this point are described from an application domain perspective, and can be expressed by the user. Examples of requirements at the SaaS layer can be a web application's response time, the maximum desired price for the execution of a set of batch jobs, or the level of security required for the application's communication.

Translation between the SaaS and PaaS layer begins with the user request and produces a definition understandable by the platform layer. This implies that some domain specific translation needs to take place, for example to convert execution time or price requirements to number of tasks. Performance prediction models can be used to determine how tasks can be parallelized; workflows can be generated to ensure that execution deadlines are met.

Translation to the IaaS layer map into instantiated VMs with the appropriate image software so the PaaS layer can execute on top of it; also, task mapping decisions need to be made in order to accomplish the original requests from the user.

5.2.3 Federation of Clouds

The previous section considers the communication flow inside a single provider in order to fulfill an application's request. Information is passed down across the stack to realize the contracts among layers and translate higher layer restrictions to the actual resources executing the request. However, this scenario only holds if infinite resources are assumed on a single site. Since this is not the case, providers need to collaborate to be able to fulfill requests during peak demands and negotiate the use of idle resources with other peers. This is the goal of federation.

Rather than only considering federation as a matter of two heterogeneous sites, we propose an approach in which it is defined for concrete layers with analogous capabilities. This mode of communication allows inter-site negotiation at common grounds for well understood protocols and policies. One of the benefits of this model is that we can assume that not all providers implement every layer, and therefore multiple service suppliers can be joined to fulfill one single application request. This consideration also results in the possibility of specialized, single layer providers that can be leveraged by other sites. In fact, a site does not need to provide a full implementation of a layer to be able to be part of a federation: it only needs to be able to "speak" the appropriate protocol. No assumptions are made about how a service is fulfilled, or what additional layers are involved in realizing the agreement. Some providers may internally require of other layers to complete the request, although that is not part of the federation process.

This approach is akin to the well-established open systems protocol stack model of OSI [Zim81], where different protocols are employed at each layer to implement a concrete functionality. Communication can happen between two identical layers, or between consecutive layers (either lower or higher in the stack). In the context of the cloud, we have identified the layers already discussed —SaaS, PaaS and IaaS— as the building blocks for federation. Different communication models among layers have consequent implications, that we analyze next.

5.3 Inter-Layer Delegation

In previous sections we describe our focus on the three service layers of the Cloud. We also observe that not all Cloud providers would support services at all the SaaS, PaaS and IaaS layers. Therefore, they usually only expose the service interfaces of the layers that they support. For example, when a Cloud provider supports only services at the upper level (i.e., SaaS), it rarely exposes the delegation protocols beneath said layer. It is also possible that there is no layer separation in providers' implementations. Furthermore, all service interfaces are currently provider-specific and standardization is yet to be matured and adopted by the Cloud community. There are multiple organizations which have standard activities that mainly focus on the IaaS layer. One effort is the OCCI-WorkGroup [OCC]. Another effort is the Distributed Management Task Force (DMTF), whose activities include defining the open Cloud architecture[1] and describing some exemplary use cases[2]. Participant vendors of DMTF may submit their standard specifications and reference implementation for evaluation (e.g., Oracle's resource model). However, these two set of standard activities do not yet address the protocols at the SaaS and PaaS layers.

As an open collaborative effort, the Reservoir project [RBL+09] provides their service designs between layers as well as exemplary information flow. In their model, a "service manifesto" is created by the Service layer, the Service Manager, and passed to the lower layers. The transformation of an application service requirement is done at the Service layer into detailed platform specific requirements (middleware packages for application execution platform) and also infrastructure information (CPU, memory, disk, network, etc). The delegation between layers has defined

[1]http://dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf

[2]http://dmtf.org/sites/default/files/standards/documents/DSP-IS0103_1.0.0.pdf

specific interfaces as the Service Management Interface (SMI), VEE Management Interface (VMI) and VEE Host Interface (VHI).

5.4 Federation of Clouds

As in traditional scheduling, where most systems try to achieve the best trade-off between the users' demands and the system policies and objectives, there are conflicting performance goals between the end users and the cloud providers. While users focus on optimizing the performance of a single application or workflow, such as application throughput and user perceived response time, cloud providers aim to obtain the best system throughput, use resources efficiently, or consume less energy. Efficient brokering policies will try to satisfy the user requirements and clouds' global performance at the same time. Thereby, cloud federation introduces new avenues of research into brokering policies such as those techniques based on ensuring the required QoS level (e,g., through advance reservation techniques) or those aiming at optimizing the energy efficiency. Furthermore, the layered service model proposed in this chapter enables the isolation between brokering policies in federated clouds at different layers which can be implemented following different approaches.

Existing work in cloud brokering focuses on the federation of clouds mainly at the IaaS layer such as those strategies based on match-making on top of clouds [CTVP10], advanced reservations [SMLF08, SKF08] or energy efficiency [BBA10].

Table 5.1 summarizes the main objectives and parameters of brokering at different layers of cloud federation. Some objectives, such as cost-effectiveness, are desired across all layers, though with different pricing methods. In the following subsections we discuss in more detail the possibilities and characteristics of brokering at different layers of federation, starting from SaaS layer that is the closest layer to the users.

	Parameters	Objectives
SaaS	User requirements Service level agreements Software licensing	Maximize QoS delivered Minimize cost Functionality/availability
PaaS	Compiling requirements Runtime requirements Runtime licensing	Functionality Optimize applications' execution Fault tolerance
IaaS	Resource characteristics Monitoring data (hardware/VMs) Modeling/benchmarking data Constraints/requirements (deadline, budget, etc.)	Maximize cost-effectiveness Acceleration Conservation Resiliency Maximize energy efficiency

Table 5.1: Summary of brokering goals at different layers of cloud federation

5.4.1 Brokering at the SaaS Layer

Brokering at the SaaS layer is mainly based on the user's requirements and Service Level Agreements (SLA) between different cloud providers. As mentioned in section 5.2, a cloud provider that implements the SaaS layers should guarantee a given level of service for a set of application requirements. The application's requirements can be generic and/or specific. Generic requirements do not depend on the characteristics of the application and can be used for many types of applications. Some examples are: response time (or completion time), cost (of running the application), and level of security. Specific requirements deal with the characteristics and input parameters of the application. Taking WRF as a use case, some specific application requirements are: application version, geographic region, or resolution of the simulation.

Table 5.1 overviews the main objectives of brokering in the SaaS federated cloud, which are on three different dimensions: QoS, cost and functionality/availability. However, the actual brokering policies should address more concrete objectives that would consider different goals and also both generic and specific application requirements/input parameters. Possible optimization goals include:

- Using only generic application requirements: lowest price for a given completion time, shortest completion time for a given budget, highest security level for a given budget
- Using both generic and specific applications requirements (WRF): shortest completion time for a given simulation resolution, higher simulation resolution for a given budget and completion time

In order to achieve the objectives listed above, federated clouds will have to handle and exchange information at the SaaS layer such as: estimated application completion time, cost of running the application, cost of software licenses, available software/versions or limitations (e.g., for the use case of WRF, the maximum simulation resolution). Based on information and the objectives described above, different strategies can be considered. Some examples are:

- Forwarding: if the originator cloud cannot accommodate the request or another cloud can provide better cost-effectiveness, the request can be forwarded to another cloud domain of the federated cloud. Benchmarking or modeling the applications on the clouds' resources may be used to estimate the cost or completion time for a given application, but this is a transparent process at the federation level (each cloud can have its own mechanisms).
- Negotiation: one cloud may take care of jobs from another cloud upon agreement. The negotiation can be based on information from both past and future events. For example, a job request might be forwarded to a cloud at higher cost but doing so may significantly optimize the energy efficiency (e.g., switching down servers and/or CRAC units). Other considerations could be taken

into account during the negotiation such as the cloud reputation (e.g., based on SLA violation rate).

5.4.2 Brokering at the PaaS Layer

Brokering at the PaaS layer is mainly based on the application's requirements in terms of deployment (e.g., compiler framework) and runtime support (e.g., libraries). Since compiling tools, libraries and runtime environments can be from different vendors and with different characteristics, they can have different licensing conditions, prices and even different functionality and performance. Furthermore, additional characteristics such as fault tolerance or platform security issues can be considered in brokering policies at the PaaS layer. Given the use case of WRF the parameters are based on MPI, such as the MPI compiler characteristics, runtime environment for MPI applications and their associated costs and limitations (e.g., licenses for specific MPI runtime).

The main goals of brokering a federated cloud at the PaaS layer are focused on improving the the applications' environments, including:

- Functionality/availability: brokering over multiple clouds increases the probability of provisioning with more specialized compilers or execution environments.
- Optimize applications: in some sense, the objective is maximizing the potential of the applications to obtain better performance. Policies can decide using specific compilers or runtime in order to obtain, for example, more efficient binaries for a given cloud.
- Fault tolerance and security: when choosing a specific execution environment from different clouds, fault tolerance and security are attractive secondary

goals that may add value to a given decision or they can be primary goals if the nature of the application requires of them.

In order to meet the objectives such as those described above, different clouds must handle and exchange information related to the compiling frameworks such as vendor, capabilities, versions, compatibility or licensing costs, and information related to the runtime characteristics and limitations such as MPI implementation version, vendor, specific libraries or number of MPI processes supported.

Brokering policies at the PaaS layer will try to find the best trade off between the optimization goals discussed above and the limitations from the other layers such as the cost. As a matter of example, a brokering policy may decide to compile the WRF application using an expensive compiling framework if the possible optimizations may result in lower completion time in the associated execution framework. Also, the decision can be using a higher number of MPI processes in order to maintain the QoS delivered to the users.

5.4.3 Brokering at the IaaS Layer

When addressing federation at the IaaS layer, we consider cloud infrastructures to be hybrid, integrating different types of resource classes such as public and private clouds from distributed locations. As the infrastructure is dynamic and can contain a wide array of resource classes with different characteristics and capabilities, it is important to be able to dynamically provision the appropriate mix of resources based on the objectives and requirements of the application. Furthermore, application requirements and resource state may change, for example, due to workload surges, system failures or emergency system maintenance, and as a result, it is necessary to adapt the provisioning to match these changes in resource and application workload.

Brokering functions in federated clouds at the IaaS layer can be decomposed into two aspects: resource provisioning and resource adaptation. In resource provisioning, the most appropriate mix of resource classes and the number of nodes of each resource class are estimated so as to match the requirements of the application and to ensure that the user objectives (e.g., throughput) and constraints (e.g., precision) are satisfied. Note that re-provisioning can be expensive in terms of time and other costs, and as a result, identifying the best possible initial provisioning is important. For example, if the initial estimate of required resources is not sufficient, additional nodes can be launched. However, this would involve additional delays due to, for example, time spent to create and configure new instances. At runtime, delays can be caused by, for example, failures, premature job termination, performance fluctuation, performance degradation due to increasing user requests, etc. As a result, it is necessary to continuously monitor the application execution and adapt resources to ensure that user objectives and constraints are satisfied. Resource adaption is, therefore, responsible for provisioning resources dynamically and on runtime. Examples are assigning more physical CPUs to a given VM to speed up an application, or migrating VMs in order to reduce the resource sharing or optimize the energy efficiency.

The goals of brokering methods and policies in federated clouds at the IaaS layer can be found in different domains. Some examples are listed as follows:

- Cost-effectiveness: federated clouds provide a larger amount of resources, which may help improve cost-effectiveness. This include improvement for both the user and the provider such as, for a given cost, reducing the time to completion, increasing the system throughput or optimizing the resource utilization
- Acceleration: federated clouds can be used as accelerators to reduce the application time to completion by, for example, using cloud resources to exploit

an additional level of parallelism by offloading appropriate tasks to cloud resources, given budget constraints.

- Conservation: federated clouds can be used to conserve allocations, within the appropriate runtime and budget constraints.
- Resilience: federated clouds can be used to handle unexpected situations such as an unanticipated downtime, inadequate allocations or failures of working nodes. Additional cloud resources can be requested to alleviate the impact of the unexpected situations and meet user objectives.
- Energy efficiency: federated clouds can facilitate optimizing the energy efficiency of clouds by, for example, workload consolidation, thermal-aware placement or delegating part of the workload to external clouds in order to optimize the energy-efficiency of a given cloud.

Multiple objectives can be combined as needed. An obvious example is combining an acceleration objective with a resilience objective. Different kinds of information will be required to be handled and exchanged across different clouds in order to implement brokering policies with the aim of meeting the objectives presented above on top of a cloud federation. At the IaaS level the information is lower level and include:

- Monitoring information from the hardware/OS: includes hardware and OS characteristics (static information, such as CPU vendor or OS type) and dynamic information such as CPU load, CPU frequency, RAM memory utilization, free storage, type/quality of the interconnection networks (e.g., bandwidth and latency). Monitoring systems may also provide measures of power dissipated or even sensing information from the environment such as temperature or airflow.

- VM information: includes information related to the virtualization level such as hypervisor type, available VM classes, number of running VMs, and characteristics of the VMs (e.g., memory assigned to VMs, number of virtual CPUs or CPU affinity).
- Application benchmarking/modeling: it is responsible for estimating important metrics such as execution time or required number of VMs for the application. Since it depends on the actual execution platform this will be exchanged across different clouds.
- Cost: includes the costs for provisioning and VM allocation (e.g., the cost of a VM/hour, server/hour or a set of resources/hour) or data transfer cost (e.g., GB transferred).
- Other information such as data locality (e.g., VM images or actual user data) or security issues can be useful for implementing policies at the IaaS layer.

Brokering policies make decisions during resource provisioning and resource adaptation depending on the user objectives as well as information exchanged between clouds, as described above, and on the metrics used. Considering WRF as a use case, important metrics for policies are, for example, deadline and budget. For the deadline metric, the brokering decision is to select the fastest resource class for each task and to decide the cloud and the number of nodes per resource class based on the deadline. When an application needs to be completed as soon as possible, regardless of cost and budget, the largest useful number of nodes can be allocated in the cloud(s) that estimate shortest completion time. This estimation is usually based on representative benchmarking or modeling on all resource classes from all clouds. If the budget metric is enforced on the application, the type and number of allocatable nodes is restricted by the budget. If the budget is violated with the fastest resource class from the different clouds, then the next fastest and cheaper resource class is selected until the expected cost falls within the budget limit. After the initial resource provisioning, the allocated resources and tasks are monitored. The framework continually updates the metrics used by the brokering policies. If the user objective might be violated (for example, the updated cost is larger than the initially estimated cost), then additional, possibly different, resources will be provisioned and the remaining tasks will be rescheduled.

Cloud federation at the IaaS layer offers many opportunities for energy optimization, which is another important metric that is becoming crucial in large-scale distributed system such as clouds. Different techniques such as VM migration combined with switching on/off servers or workload consolidation/placement (including thermal-aware approaches) can be used for brokering since the resources belong to multiple clouds that may be in different operational states. Even techniques such as DVFS can take advantage cloud federation when, for example, cloud providers' policies are not exceeding a given peak of power dissipated or energy consumed. Furthermore, the price differences of the electricity based on the geographical location and time during the day can be leveraged to implement energy-aware polices or even the source/class of electricity used for the clouds [LBN$^+$10] can be taken into account in order to implement environmental-friendly policies.

5.5 Weather Research and Forecasting (WRF) as a Service

We present the WRF application [MDG$^+$04] as a use case for the federated cloud architecture of Section 5.2. WRF is parallel scientific application which performs mesoscale weather simulations of user-selectable geographic areas, with a given resolution for each area. Due to the nature of certain weather phenomena such as hurricanes or tornadoes, performing accurate predictions in very short time spans is vital to make appropriate preparations involving business operations management and government and human related logistics. Thus, sharing of resources between institutions to provide elasticity and dynamic capacity in extreme situations is key.

WRF benefits from a hosted service architecture since it is a cross-domain application, requiring extensive IT administration and setup expertise in addition to scientific and meteorological knowledge to run it. Establishing WRF as a SaaS using the layer model separates the concerns of the scientists from the underlying platform and infrastructure issues. Efforts to separate these domains of expertise are ongoing, and at the service level a web portal has been developed as one approach[3] to hide IT concerns from users.

The subsequent sections show how the architecture of Section 5.2 is applied to support federation and delegation while separating the WRF end user concerns from those of the compute layer software and infrastructure at the PaaS and IaaS layers. Table 5.2 summarizes the key application parameters at each layer as discussed below. As alluded to earlier, specific details about the implementation of the translation is beyond the scope of this work. While the implemenation of such a translation mechansim specifically for WRF would not be too complex, a translator would need to account for different kinds of goals. For example, a web application service needs to reliably service a certain number of requests per unit time, whereas a scientific application like WRF needs to finish executing an entire program before a given deadline. It would also need to distinguish between soft and hard deadlines.

[3]http://www.wrfportal.org

Layer	Input parameters	Transformation from upper layer	Output from lower layer
SaaS	Region data files, software version, number of parallel runs, deadline, budget	—	Total execution cost, Total execution time
PaaS	Number of tasks, software packages	Prediction model to calculate number of tasks, list of required software packages	VM execution costs, VM execution time
IaaS	Number of VMs, VM image (OS, filesystem)	Mapping of tasks to VMs, VM image handles, VM parameters	—

Table 5.2: WRF as a Service workflow

5.5.1 Software as a Service Layer

We propose a SaaS solution where users can request WRF executions by providing high level requirements. When these requirements are entered via a GUI on a web portal the portal generates underlying files that are needed by the WRF executable. An example are the region files that contain geographic and weather related data.

The input parameters are:

- *Input files:* Files that need to be processed during the experiment. These include a namelist file and its corresponding region files. The namelist file specifies all the runtime options desired by the user. The region files are binary files that describe the geographical area.
- *WRF version:* Users may need results for a specific version of the software
- *Parallel executions:* How many ensemble runs to execute in parallel. (The service allows users to specify ensemble runs, where the multiple experiments

on the data are executed, but with different inputs. In the end, the results from all runs are averaged. This may achieve more accurate results.)

- *Deadline:* When should the experiment finish
- *Cost:* How much is the user willing to pay for the service

The user specifies the listed parameters to define the execution of the tasks without needing to consider PaaS and IaaS details such as machine architecture or virtualization platform.

For inter-cloud federation the SaaS layer implementation has the option to forward this input to a partner WRF SaaS layer provider accepting these parameters and files. The decision whether to 'sub-contract' this particular job to a partner is based on a policy with one or more of the considerations presented in Section 5.4. Also note:

- The target provider must be able to access the experiment's input files, either by transferring them or retrieving them from a catalog
- In the case of an ensemble run (when the number of *parallel executions* is higher than 1), a site may choose to transfer one or more instances of the experiment, given that the total cost is not higher than the cost defined by the user and that no instance will fail to satisfy the deadline

Instead of federating, the SaaS layer can delegate the execution to to the PaaS layer. In order to transfer control down in the stack, the user's request is translated to PaaS layer parameters as discussed next.

5.5.2 Platform as a Service Layer

In our architecture, the PaaS layer is constructed by wrapping the MPI libraries and making them available as a service. The motivation for this is discussed in section 5.2.1. Additionally, the PaaS layer is in charge of providing and managing the middleware that allows execution of WRF. In this case, we consider the following items as part of this layer:

- *WRF executables and required libraries:* The PaaS layer needs to ensure that the required software will be available at the provider side. It needs to guarantee that the required operating system and appropriate library versions can be accessed at the site.
- *Software licenses:* In the case of libraries or software that requires licenses, such as certain compilers or operating systems, the PaaS layer needs to certify that the required number of them will be available during execution.
- *Task decomposition:* Another job of this layer is to manage MPI execution, in terms of running the appropriate number of tasks to meet higher level requirements. The user that interacts with the SaaS interface does not need to specify how the experiment has to be decomposed in tasks, but that mapping needs to be resolved at the middleware management level.

Delegation from the SaaS layer to the PaaS layer needs to be managed by a translator that ensures the original request objectives are maintained. Some input criteria need to be converted to the appropriate input for this layer, while others are passed down the stack. A key challenge in this case is satisfying the quality of service or completion deadline requirement. We consider the use of a prediction model in order to calculate possible costs and task decomposition that can meet the requested deadline. However, in a federation of clouds, the compute resources are heterogeneous, so the predictor needs to be able to determine performance numbers for different combinations of resource requirements. The predictor needs to determine the resources needed given execution deadline, cost, and application input parameters. *Aprof* [SFB$^+$08] is an example of a predictor capable of calculating runtime values in a cluster environment given a list of arbitrary resource requirements. A key difference between the usage of a prediction model for traditional cluster computing and in the federated cloud model we propose in this work is that the resource selection process is more complex in the latter. This is because there can be a much larger pool of heterogeneous resources. Also, there can be competing constraints, such as time, cost, and availability. In this case, we can consider a couple of options. One is to use a feedback mechanism in which users are given different options, e.g. different costs for different execution times that satisfy the time and cost constraints. The other is to give priorities to different constraints. For example, a user may not mind waiting slightly longer for a program to complete as long as it finishes before the deadline, so they will give priority to cost, such that the amount they spend is minimized. An important consideration for users with hard deadlines is that the error of the prediction model must be accounted for. This is particularly true for statistical models that rely on historical execution data, which are desirable in our case for their performance, but tend to have prediction error ranging from 5-30%, depending on the application, run time configuration, number of available data points, and number of parameters being modeled.

The PaaS layer receives the number of desired tasks from the prediction model used to translate user requirements to this layer's input. Figure 5.4 shows some of the run times of WRF under different execution parameters, according to the *aprof* predictor. Using this tool, we can determine the approximate time of the requested execution for different types of resources. The figure shows two predicted systems,

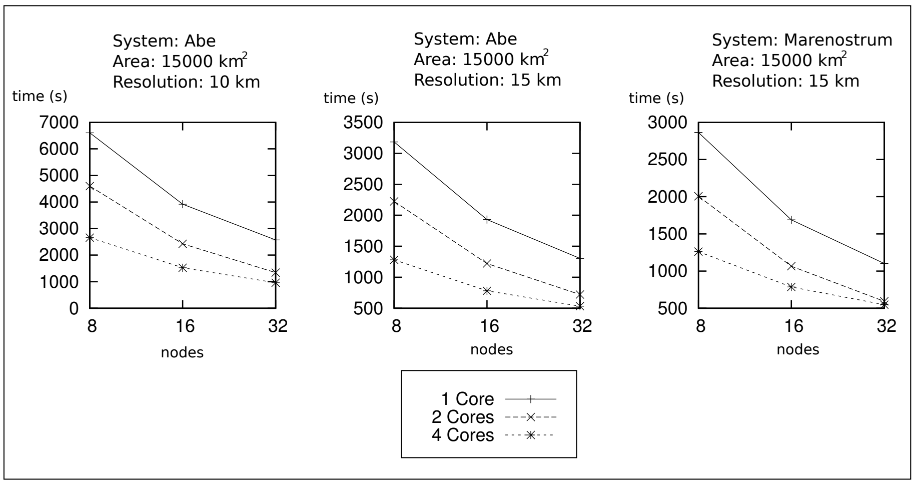


Figure 5.4: Prediction models for different areas and resolutions

which have the same characteristics as Abe, one of Teragrid's[4] computing clusters, and Marenostrum, the supercomputer at *Barcelona Supercomputing Center*. The plots were generated for simulation areas of 15000 square kilometers with resolution of 10 and 15 kms. This data can be used to calculate the approximate execution time of a WRF request for any number of systems.

The second task of the translation from SaaS to PaaS layer is to compile a list of required software packages and operating system images that will be requested when the execution VM is instantiated. This is accomplished by using a mapping from the user input to a set of predefined software (*e.g.*, Linux kernel version, Fortran compilers and runtime libraries and MPI version). Additionally, the translator needs to account for the appropriate licenses needed to run the required software.

Again, the PaaS layer may decide to either off-load the work to another peer through an appropriate federation protocol, or fulfill the request with local resources by delegating it to the IaaS layer, in which case the request needs to be translated to the corresponding input values.

[4]http://www.teragrid.org

5.5.3 Infrastructure as a Service Layer

The IaaS layer provisions the execution environment to run the application. This layer's interface needs to publish which resources it supports and the associated cost. Also, the IaaS component needs to consider staging-in of data and application binaries —*e.g.*, in the form of Virtual Machine images.

Delegation from the PaaS layer again needs to happen through a translation component. First, the different combinations of resources produced by the prediction model are compared with what the virtualization manager can provide to calculate execution costs, then those parameters (amount of RAM, number of virtual processors, etc.) are passed to the IaaS manager to be used during VM instantiation. Next, the list of software needs to be retrieved by the IaaS layer to provision the VMs. There are different methods to do this, one example would be by associating a virtual disk image located in a file repository with the list of software components; another example would require creating the virtual disk image on demand before execution by aggregating the software packages from a repository.

Once the resources have been provisioned, the IaaS layer instantiates the required VMs and control is given back to the PaaS component, which orchestrates the provisioning of VMs and the execution of the software on them. The platform layer is in charge of issuing MPI calls to define which virtual hosts will take place in the execution, spawning the required number of processes, and ensuring the application is run successfully.

However, in the cases where the infrastructure layer does not have the necessary resources, or when the site's policies mandate it, the request issued by the platform component can be forwarded to another site via a federation protocol. In this case, IaaS providers need to be able to publish their resources, to which disk image repositories they have access and their execution costs. Based on the user's requirements, the site may acquire external resources to answer a request after employing the federation protocol.

5.6 Conclusion and Future Work

In this chapter we have presented an general approach to the cloud federation problem by considering a layered model where negotiation is constrained to well-defined sets of parameters. We have discussed the benefits of decoupling the different layers —Infrastructure, Platform and Software as a Service— so that the execution of an application can be supported by diverse providers implementing different parts of the functionality. Additionally, we explain how user and site policies can be used to negotiate federation between partners, or translated to delegate tasks to other layers of a single site.

We have also introduced a motivational scenario to illustrate this layered model. We described a "WRF as a service" application for domain experts which accepts high level parameters relating to user requirements such as cost or time of execution. We have showed how these requirements are either used in the negotiation process or transformed to new arguments to lower levels in the cloud stack by using prediction models and inter-layer translation mechanisms. Finally, we have discussed different brokering strategies for providers to consider assigning parts of the execution workflow to other partners while enforcing user policies.

As a remaining contribution for this work, we propose an implementation of the model's lower two layers. The next chapter presents how the previously discussed work can be leveraged to enable the distribution of jobs and sharing of resources among providers.

CHAPTER 6

AN ANALYSIS OF PROVISIONING AND ALLOCATION POLICIES FOR INFRASTRUCTURE-AS-A-SERVICE CLOUDS

In the previous two chapters, we have defined a meta-scheduler to execute jobs across organizations' physical resources and an Infrastructure as a Service manager to instantiate virtual machines. However, acquiring virtual infrastructure and mapping jobs to it requires different strategies that need to be studied jointly. In this chapter we identify the two stages involved in the process of executing job workloads on virtual resources, namely job allocation and VM provisioning, and experimentally investigate their behavior based on metrics such as cost or makespan. We provide a taxonomy for both types of policies and analyze them in real clouds at FIU, TU Delft and Amazon EC2 by executing synthetic workloads representing different arrival and execution patterns. Finally we explore the interplay between allocation and provisioning and propose a set of coordinated policies that result in lower costs and job slowdown.

6.1 Introduction

Recent advances [GWT+08, GHJ+09] in the high-speed yet low-cost interconnection of off-the-shelf computational and storage resources have facilitated the creation of data centers of unprecedented scale. As a consequence, a fundamental shift is beginning to occur in the way computational resources are provisioned and allocated by our society, from traditional ownership to Infrastructure-as-a-Service (IaaS) clouds—leasing and releasing virtualized resources.

Although hundreds of commercial IaaS providers exist, to transition to IaaS clouds users still need detailed understanding of achieved performance and incurred cost. In particular, potential IaaS users need to understand the performance and cost of resource provisioning and allocation policies, and the interplay between them. To address this need, we conduct an empirical analysis of resource provisioning and allocation policies for IaaS clouds.

The basic operation of an IaaS cloud is to temporarily instantiate on-demand virtual machines (VMs)—of pre-agreed computing power and memory size, operating system and provided libraries, and, usually, some measure of Quality of Service (QoS). Providers host the resources shared by all users and can take advantage of economies of scale by leasing their physical infrastructure as virtualized resources to many different classes of users. Users provision, that is, acquire and release, resources based on their actual needs, only when, where, and for how long needed; they can allocate the provisioned resources according to the specific requirements of the workloads at hand. This model makes IaaS clouds a flexible solution that reduces or completely eliminates the need for acquiring and managing costly physical resources, but also introduces the need to consider *online* the trade-off between performance (more resources) and cost.

The provisioning and allocation policies can have an important impact on the traditional performance metrics, from workload makespan to individual job slowdown [dAdCB09]. Since instantiating a large number of VMs is simple, over-provisioning can incur a substantial yet unnecessary cost; when the allocation policy is inefficient, a dynamic provisioning policy may lease resources that remain largely unused [GG11]. The pricing scheme of the IaaS cloud, which may include hourly charging periods and discounts for first-time use, may lead to different cost gains than expected from actual resource consumption.

The performance-cost trade-off has been extensively studied in the context of grids, mainly from the perspective of users also acting as providers and looking for economic gain [SAL$^+$04] or for sharing fairness [BAGS02]. Moreover, more traditional resource managers such as Condor support [TTL05, DSL+08], through versatile job specification languages, complex criteria for the selection of resources. Several studies [dAdCB09, KV10, GG11] have approached this trade-off in simulation, in the context of clouds. However, until now no study has investigated in practice the impact of the provisioning and allocation policies that users can employ, and of the interplay between these policies, in the context of clouds. The need for empirical evaluation stems from recent results [JPC09, IYE11] in cloud performance evaluation, which show that cloud performance is much lower and more variable than considered in simulation models. In this chapter we propose a comprehensive investigation of provisioning and allocation policies for IaaS clouds. Our main contribution is threefold:

1. We identify eight provisioning and four allocation policies that can realistically be applied for managing workloads in IaaS clouds (Section 6.3);
2. We conduct an empirical investigation using three IaaS clouds, including the services provided by the commercial IaaS Amazon EC2 (Section 6.4);
3. We analyze empirically and, only for the otherwise expensive experiments, in simulation the performance and cost of resource provisioning and allocation policies, and the interplay between these two types of policies (Section 6.5).

6.2 System Model

In this section we present the system model used throughout this work.

6.2.1 Workload Model

Although desirable, it is not yet possible to define a realistic workload for IaaS clouds, due to the scarcity of public workload traces or common practice reports. The Grid

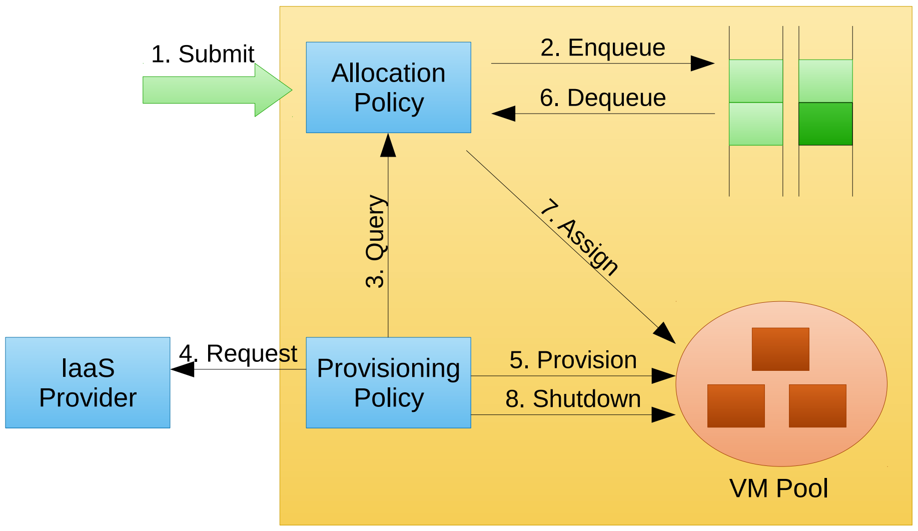


Figure 6.1: The cloud ecosystem.

and Parallel Workload Archives provide in total tens of workload traces, but it is yet unknown if the users of these environments will migrate to IaaS clouds [IOY+11]. Iosup et al. have already shown in a characterization of over fifteen grid workloads [IE11] two trends: the disappearance of tightly-coupled parallel jobs in favor of Bags of Tasks (BoTs), and the decrease of the amount of work (runtime) of each task.

In this chapter, we consider synthetic, BoT-based workloads with runtimes typical for data mining and semi-interactive processing, that is, several minutes [HKZ+11, CGGK11]. In our workload model, jobs are CPU-bound and their runtime is dependent on the speed of the (virtual) processor where they are executed.

6.2.2 Resource Model

In our system model, resources are provisioned exclusively from IaaS clouds. Although hybrid local-cloud systems still exist, we anticipate with this work the moment when buying and maintaining local resources will be anachronistic. The cloud ecosystem investigated in this work, which is comprised of the IaaS cloud provider(s) and user(s), is depicted in Figure 6.1. We assume that users send their workloads to a scheduler, which enqueues jobs and assigns them to the pool of available resources based on an *allocation policy.* A system component manages the pool of resources via a *provisioning policy*, that is, a policy that decides when to lease and to release resources from IaaS clouds. This component can query the state of the allocation, such as the queue size or the average waiting time for jobs.

We model the operation of IaaS clouds based on Amazon EC2, a popular commercial IaaS. We assume that a provisioning request issued to the IaaS cloud will incur delays that depend on the process used by the IaaS to select, lease-and-boot, and release (shut down) a VM instance. Last, we assume that VMs have a cost associated to their operation, with a cost model that is proportional, possibly non-linearly, to the runtime of the VM instance, and not counting the time required to boot or shut down the instance.

6.3 Provisioning and Allocation Policies

We present in this section the provisioning and allocation policies considered for this work. Although we envision that future policies will adapt to changing workload, evolving resources, and complex Service Level Agreements, we focus in this work on the simple policies that may be realistically employed in practice, today.

6.3.1 Provisioning Policies

We consider for this work eight provisioning policies; we summarize their properties in Table 6.1 and describe them next. Overall, we identify two main classes of provisioning policies, static and dynamic. For the class of dynamic provisioning policies, we investigate three criteria for deciding when the policies are called (*triggers*): the

Policy	Dynamic	Trigger	Job Duration	Increase Param.
Startup	No	—	—	—
OD-S	Yes	Queue size	No	1
OD-G	Yes	Queue size	No	n
OD-ExecTime	Yes	Exec. time	Yes	1
OD-ExecAvg	Yes	Exec. time	No	1
OD-ExecKN	Yes	Exec. time	No	1
OD-Wait	Yes	Wait time	No	1
OD-2Q	Yes	Queue size	Partial	1

Table 6.1: Overview of provisioning policies.

current size of the queue, the accumulated execution time of the queued jobs, and the total waiting time among queued jobs. Some of the provisioning policies considered here require information about the duration of jobs, which is not always provided by the user; we consider, for these policies, alternatives that estimate this information from historical records. The last policy in Table 6.1 requires partial information, that is, it only requires classification information—a job can be either small or long. We now describe the provisioning policies, in turn:

1. **Startup** is a provisioning policy that leases all the needed VM instances during the system startup. While this ensures that no delay will occur due to waiting for new resources to be leased, this policy is not flexible–whether jobs arrive or not in the system, VM instances are still leased and paid; also, this policy cannot cope well with bursty system overloads.
2. **On-Demand, Single VM (OD-S)** is a naïve dynamic provisioning policy that leases a new VM instance for each job that cannot be assigned to a resource. VMs are shut down when they are not used for a certain duration, determined as a parameter; by setting this parameter to *0 seconds*, VMs are

released immediately after the job is completed. In general, this policy can lead to *thrashing*, that is, frequent leasing and releasing of VM instances.

3. **OD-Geometric (OD-G)** is a dynamic policy that extends `OD-S` with geometric provisioning, that is, when new instances are needed this policy provisions $n^0, n^1, n^2, ...$ machines in successive leases, where n is the *increase parameter*. Similarly, this policy releases increasing amounts of instances. The decision of (re)leasing instances is taken periodically, when the number of available VMs falls below a threshold, or the number of tasks in the system queue exceed a limit; here, we use a period of 20 seconds and (re)lease again whenever there is at least one queued job/idle VM instance.

4. **OD-ExecTime** is a dynamic provisioning policy that uses the future execution time of queued jobs, which is assumed to be known *a priori*, to decide on leasing or releasing VM instances. The decision to lease is adaptive to the cloud, in that the execution time of queued jobs must exceed the average time needed to provision and boot a VM instance (as observed for previously leased VMs) by a specified factor, which is set in this work to 5.

5. **OD-ExecAvg** is similar to `OD-ExecTime`, but the execution time for each queued job is *estimated* as the average execution time of all the jobs that have already completed. An initial prediction of the average job run-time must be provided by the user.

6. **OD-ExecKN** is similar to `OD-ExecAvg`, but uses a predictor based on [IÖF96]. For each job in the queue, `OD-ExecKN` acquires its k-nearest neighbors based on the job input parameter size. Then, the estimated execution time for a job is the average over this set of k neighbors.

Policy	Uses Queue	Job Duration	Provisioning-Aware
FCFS	Yes	No	No
FCFS-NW	No	No	No
SJF	Yes	Yes	No
FCFS-2Q	Yes (2)	Partial	Yes

Table 6.2: Overview of allocation policies.

7. **OD-Wait** is similar to `OD-ExecTime`, but it considers the waiting times of queued jobs instead of their future execution time.

8. **OD-2Q** is a provisioning policy that works in conjunction with the `FCFS-2Q` allocation policy (Section 6.3.2). To minimize the trashing of VMs for short running jobs, we define a bi-queue on-demand provisioning policy that leases VM instances and assigns them to one of two pools. The two pools effectively implement two `OD-S` queues with separate idle time parameters; here, we use longer idle times for VMs that will run short jobs, so that thrashing is reduced.

6.3.2 Allocation Policies

We consider for this work four allocation policies; their properties are summarized in Table 6.2. Most policies we investigate use one queue, but we also consider policies that use two or no queue. Similarly to our approach for provisioning policies, we consider here allocation policies that may require (partial) information about the job duration. The last policy in Table 6.2 is *provisioning-aware*, that is, it works in conjunction with the `OD-2Q` provisioning policy. We now describe the allocation policies, in turn:

1. **First-Come, First-Served** (`FCFS`) is a traditional allocation policy that assigns one task per resource in the order in which the tasks have been submitted to the system.

2. **FCFS-NoWait** (`FCFS-NW`) is an extension to `FCFS` where jobs are not queued when no available VMs exist. Instead, this policy assigns jobs to VMs that are already running other jobs, round robin. This policy eliminates the wait time, but may introduce bottlenecks in the execution of jobs.

3. **Shortest-Job First** (`SJF`) is a traditional allocation policy that gives priority to shorter jobs, assuming there is some information about or estimation of their actual duration. Although it alleviates the `FCFS` problem of short jobs waiting for the allocation of longer jobs with earlier arrival time, it can lead to starvation for long jobs.

4. **FCFS-MultiQueue** is an extension to `FCFS` where several `FCFS` queues, one for each range of job durations, are maintained. The simplest case considered here, `FCFS-2Q`, has two queues, one for short jobs and another for long jobs. Although an estimation of the runtime is necessary for this policy, it is enough to have partial knowledge of it to classify jobs. This policy can work in conjunction with a provisioning policy that divides VMs into pools for short and long running jobs, such as `OD-2Q` (introduced in Section 6.3.1).

6.4 Experimental Setup

In this section we discuss our experimental setup, in turn, the SkyMark empirical performance evaluation tool, the used testbeds, the workloads, and the employed metrics.

6.4.1 The SkyMark Empirical Performance Evaluation Tool

Unless otherwise specified, we have conducted the experiments presented in this work in *real* environments. To this end, we have implemented SkyMark, a performance evaluation framework that enables the generation, submission, and monitoring of complex workloads to IaaS cloud environments. SkyMark extends C-meter[YIEO09], an IaaS performance evaluation framework, with provisioning and allocation policies, new workloads, and new analytical capabilities. SkyMark currently supports IaaS clouds that implement Amazon EC2's interface, including all deployments using Eucalyptus, but interfaces to other clouds can be easily plugged-in to the tool. SkyMark also supports the XML-RPC system interface of OpenNebula.

The experimental process uses SkyMark as follows: The user provides a workload description file which is used to generate a real or synthetic workload. The workload is then submitted to a cloud, using pre-specified provisioning and allocation policies. Performance statistics are gathered throughout the execution of the workload and then stored in a database. Last, the database is used for post-experiment analysis. SkyMark effectively plays the role of both the user and the policies depicted in Figure 6.1.

Since using a real testbed is constrained by actual resource and budget availability, we have also developed a discrete event simulator that duplicates Skymark functionality. The simulator reads a workload description and generates events for job arrivals, VM lifecycle, and pluggable allocation and provisioning policies.

6.4.2 Experimental Environments

We have performed experiments on three *real* IaaS clouds with different resources, middleware, and virtualization systems. The three used systems are: the Delft cluster of DAS-4[1], a six-cluster wide-area distributed system in the Netherlands; a

[1]http://www.cs.vu.nl/das4/

System	Hardware Spec	VIM/Hypervisor	Max VMs
DAS4 Delft	8 dual quad-core 24 GB RAM	OpenNebula 3.0 / KVM (Full, HVM)	64
FIU	8 Pentium 4 5 GB Memory	OpenNebula 2.2 / XEN (Para., No HVM)	7
Amazon EC2 eu-west-1	-	- / XEN (Para., No HVM)	20

Table 6.3: Overview of the experimental environments.

cluster at Florida International University (FIU); and the Amazon EC2 commercial IaaS. The properties of the three IaaS clouds used in this work are summarized in Table 6.3.

6.4.3 Workloads

We have used the following workloads, each hour-long:

1. **Uniform:** A steady stream of tasks throughout the experiment; uses a Poisson arrival distribution. The average system load is around 70%.
2. **Increasing:** The workload intensity increases over time, in steps. The average system load is around 50%.
3. **Bursty:** The workload features short spikes of intense activity amid long periods of mild or moderate activity. The average system load is around 15%; the maximum load is, for a few minutes, 170%.

For the simulated results in section 6.5, we use the Uniform and Bursty workloads, and add a new one, Periodic, following a periodic increasing and decreasing arrival pattern. For these workloads we also change the job durations. More details are discussed in the corresponding section.

The jobs that comprise the workloads are synthetic and have an average execution time of 47 seconds with a standard deviation of 41.1, as measured on DAS-4 when jobs were run independently. The reason for selecting short job durations were discussed in Section 6.2.1.

6.4.4 Performance, Cost, and Compound Metrics

We use a variety of performance, cost, and compound metrics to analyze the impact of provisioning and allocation policies. For performance metrics, we look at the *workload makespan* (WMS) and at the *average job slowdown* (JSD), defined per job as the ratio of the actual runtime in the cloud and the runtime in a dedicated environment; the former metric is useful for BoT users, while the latter is useful for users of semi-interactive or individually-meaningful jobs. We also look at the workload speedup, measured against a single node (SU).

We use two cost metrics. The *actual cost* ($\mathtt{C}_a$) is defined as the sum of consumed resources, here, CPUtime. The *charged cost* ($\mathtt{C}_c$) is the price charged by the provider, here, using the Amazon EC2 pricing model of charging CPUtime consumption in increments of one hour.

6.5 Experimental Results

In this section we perform a set of experiments to explore the effects of different policies and their interactions on different systems and under varying conditions. We want to determine how policies perform, when it is better to use ones versus others, and which allocation and provisioning policies work better when used together.

We present here only representative results; for complete results, we refer to our technical report [VASI11].

6.5.1 Provisioning Policies

We first explore the effect of the provisioning policies. To this end, we use the same allocation policy, FCFS, coupled in turn with each one of the provisioning policies. We show the results in Figures 6.2, 6.3 and 6.4.

Figures 6.2 and 6.3 present the workload speedup (SU) and the job slowdown (JSD) respectively. Startup always achieves the best performance. OD-S has similar performance for the uniform workload, but is not as good for the variable workloads. From the threshold-based policies, QueueWait usually performs better than the rest, because it reacts faster to load variation. ExecTime and its variants have similar performance, with ExecTime usually performing better, since ExecAvg and ExecKN do not have exact job runtime information.

The charged cost (C_c) is shown in Figure 6.4. OD-S incurs the highest cost, since VMs are started and stopped reactively to individual job arrivals. The group of threshold-based policies and especially the Exec family of policies significantly reduce the cost of workload execution. The cost reduction becomes bigger for the increasing and bursty workloads.

6.5.2 Allocation Policies for Static Resources

In this experiment we want to study the performance of different allocation policies and the static provisioning policy, Startup. Resources are acquired at the beginning of the experiment, and then jobs are sent to the system.

We use the FCFS, the SJF, and the FCFS-NW allocation policies in the three testbeds. Figure 6.5 lists the results only for the job slowdown metric, since we did not observe significant differences in cost or makespan. The experiment shows that SJF gives a lower slowdown, since shorter jobs are processed first, which means

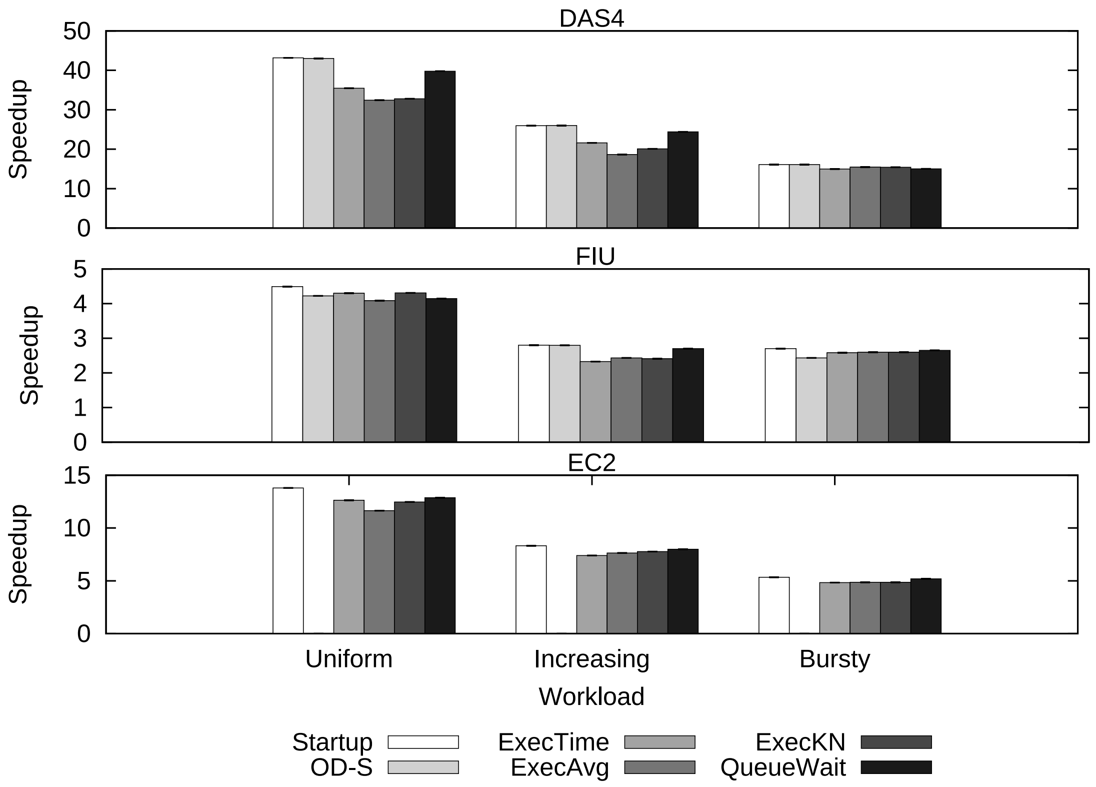


Figure 6.2: Workload speedup (`SU`) for Provisioning policies. `OD-S` was not tested on EC2, because of the significant cost that it would incur.

jobs in wait less time in the queue. Overall, `FCFS` performs similarly to SJF for the uniform and increasing workloads, however its performance degrades when under a bursty load. Lastly, the `FCFS-NW` policy, which assigns jobs to VMs with round-robin, creates resource competition, and thus has worse results in all experiments.

6.5.3 Effects of Job Size Distribution in On-Demand Policies

In this section, we investigate the impact of the job size distribution. To this end, we create an artificial workload with periodic arrival intervals with different ratios of short and long jobs. For this case, jobs have runtime averages of 10 seconds for short ones and 1 hour for long ones. We consider workloads composed of 25%, 50% and 75% short jobs (`SR25%`, `SR50%` and `SR75%`, respectively).

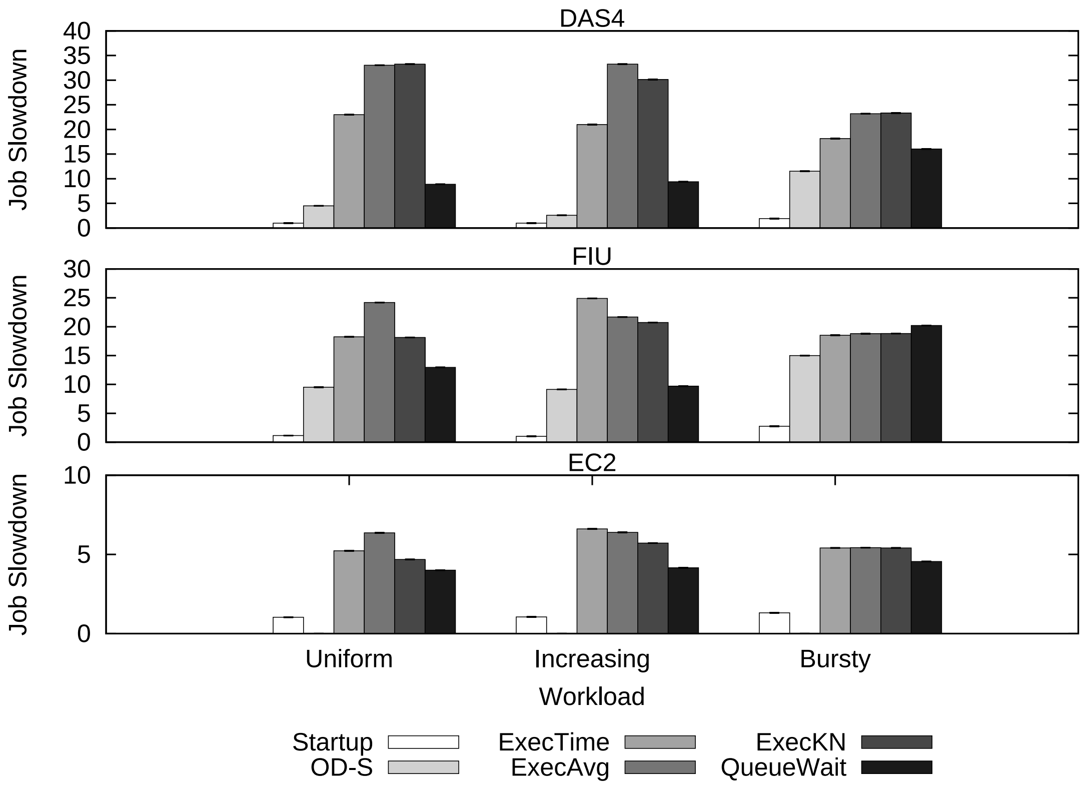


Figure 6.3: Job Slowdown (`JSD`) for provisioning policies.

As figure 6.6 shows, the proportion of short and long jobs has a high impact in the behavior of the employed policies. In particular, `SJF` favors the case when there are few short jobs, since it doesn't increase the waiting time for long running ones. However, this same policy results in much higher slowdown when the number of short jobs dominates the workload, producing starvation for long ones. Contrarily, `OD-S` with `FCFS` results in consistently high slowdown, since it doesn't offer any bias towards any type of job. Finally, our policy gives a fairer treatment to both types of jobs at the expense of fragmenting the available VMs in two groups. An additional effect of this policy can be noticed in the cost of the 75% short job ratio being slightly higher than the other two policies. This is produced by the fact that VMs for long jobs are almost immediately shut down, therefore resulting in long jobs having to wait more often. Even though this behavior doesn't have an important

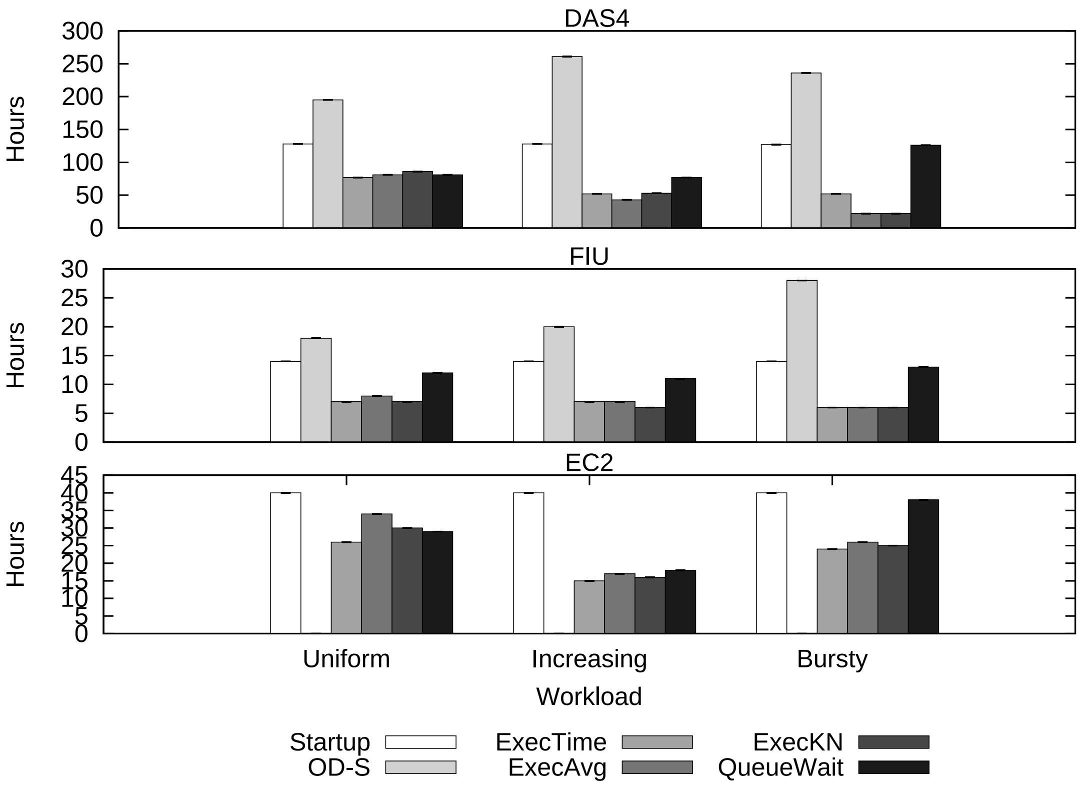


Figure 6.4: Charged cost ($\mathtt{C}_c$) for provisioning policies.

effect for the other two workloads, the case where long jobs dominate the workload makes it more noticeable.

6.5.4 Policy Interactions

We study here the interactions between allocation and provisioning policies. We use the simulator to test six pairs of policies that comprise three provisioning, `Startup`, `OD-S`, and `OD-G`; and two allocation policies, `FCFS` and `SJF`. We use three workloads with one thousand jobs where half of the jobs have a runtime average of 10 seconds and the rest of one hour. Jobs of the first workload, Uniform, have an interarrival time of 10 seconds. The second one, Periodic, has four periods of increasing and decreasing arrival times starting and ending at 60 seconds and peaking at 5 seconds.

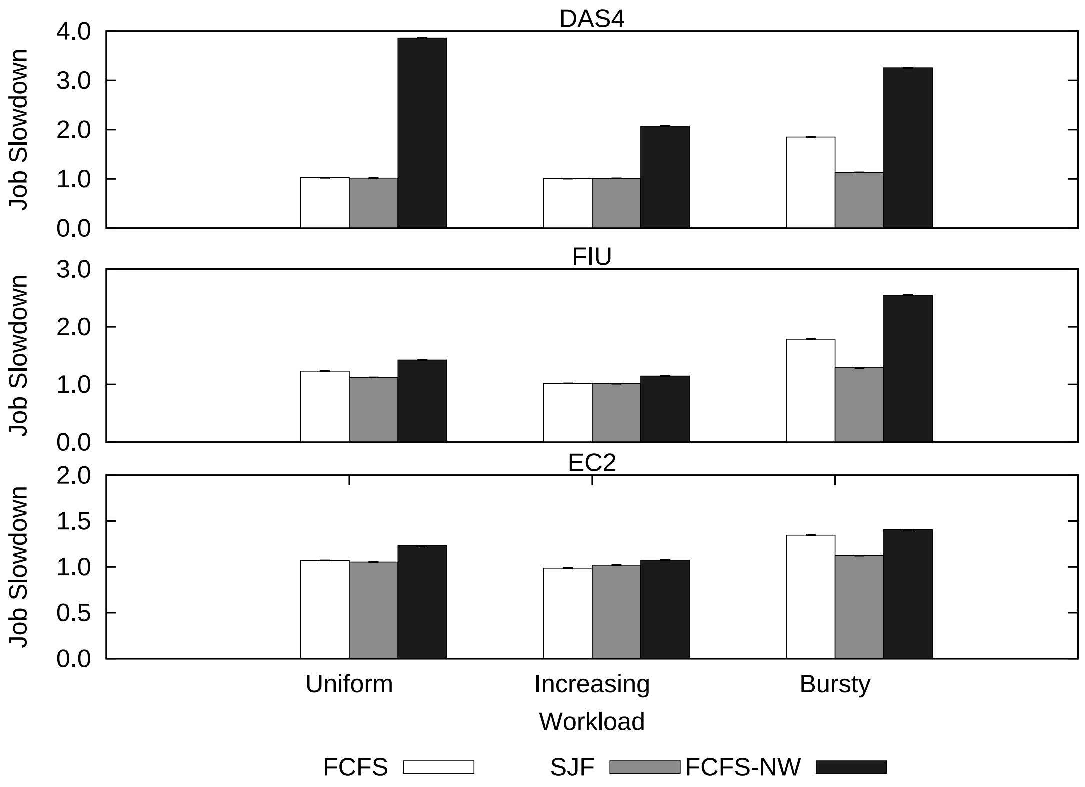


Figure 6.5: Average job slowdown for allocation policies.

Finally the last one, Bursty, alternates five periods of 200 jobs with high (30 seconds) and low (2 seconds) interarrival times.

Figure 6.7 shows the results of this experiment. We didn't include makespan in the figure since all policies had similar values. The top half of Figure 6.7 shows the cost of running the workloads, and it can be seen how on-demand provisioning policies are much more sensitive to variations in comparison to `Startup`. This is especially noticeable for the periodic workload, which has the highest variability; For the uniform workload, the system is at full utilization most of the time, minimizing the benefits of dynamic provisioning policies. The bottom half of the Figure 6.7 shows the average job slowdown, and it illustrates how the `SJF` allocation policy reduces the overall overhead for jobs by executing the shorter jobs first, and therefore minimizing the time that jobs wait in general. Another conclusion is that `Startup`

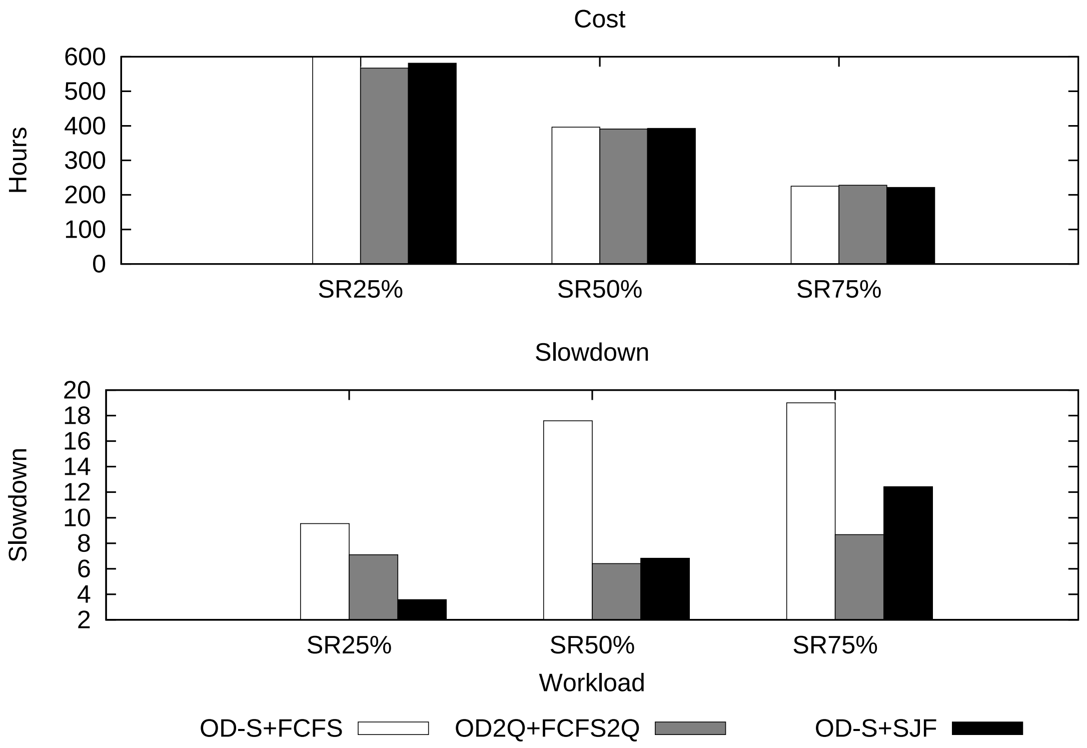


Figure 6.6: Slowdown and cost for job runtime ratios

results in lower slowdown in comparison to dynamic provisioning policies, due to the lack of overhead for VM booting and shutdown. Additionally, the figure illustrates how the `OD-G` policy results in slightly higher slowdown. The reason for this is that the geometric increase of VMs needs some time to ramp up to reach the required number of VMs to run all jobs in the queue, while the `OD-S` policy instantiates one VM for each waiting job.

6.5.5 Effects of VM Provisioning Time

In our experiments, we observe that one of the attributes that most acutely affects the behavior of policies is *VM thrashing*, already defined as the overhead produced by inefficient booting and shutdown of Virtual Machines. Thrashing is directly

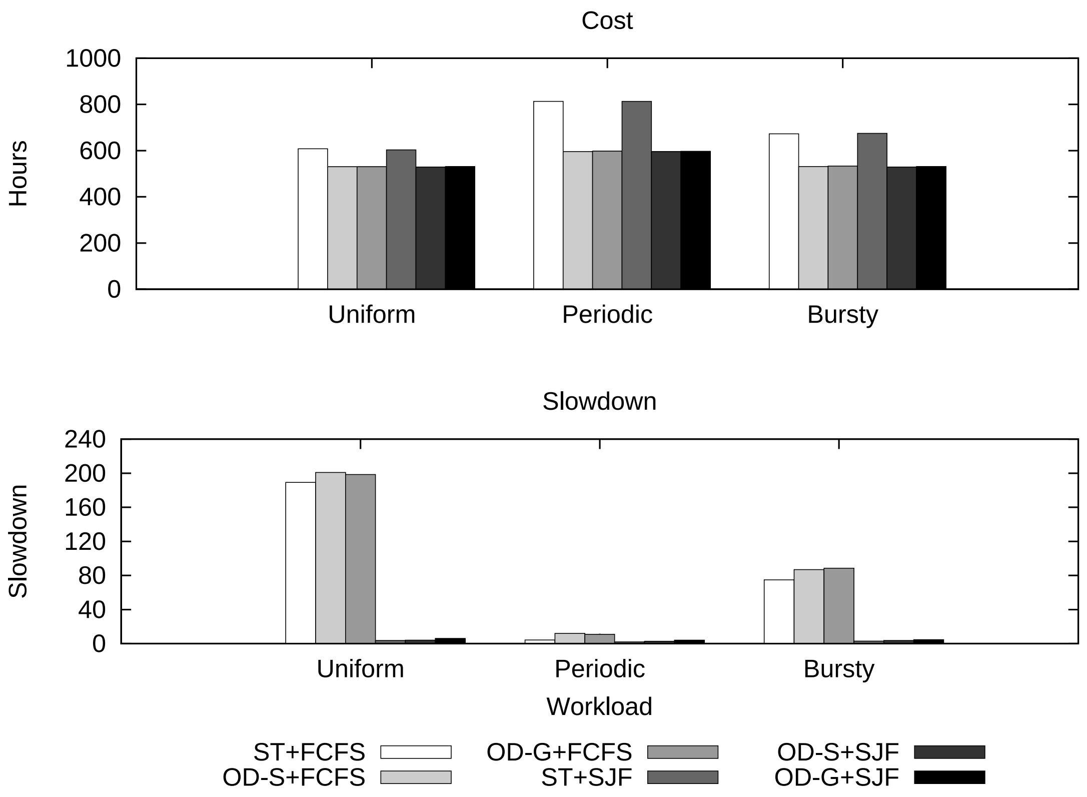


Figure 6.7: Slowdown and cost for groups of policies

related to the cost of provisioning new resources, and then destroying them, which in turn depends on the provider's VM booting time. As discussed by Iosup [IYE11], this time is highly variable for public IaaS providers such as Amazon EC2, but multiple researchers are studying better mechanisms to instantiate virtual resources, such as caching or performing virtual image snapshots. With the adoption of new provisioning improvements, we can hypothesize of their effects in existing policies to provide a glimpse of the upcoming needs in virtual provisioning systems.

One example of the effects of provisioning time in virtual providers can be seen in section 6.5.3, where the slowdown produced by high provisioning time to job execution ratio forces providers to implement reuse strategies to avoid jobs waiting for VMs to start. In our case, the pair of `OD-2Q` and `FCFS-2Q` policies take the tradeoff of fragmenting the available VMs (between a long and a short job queue) to

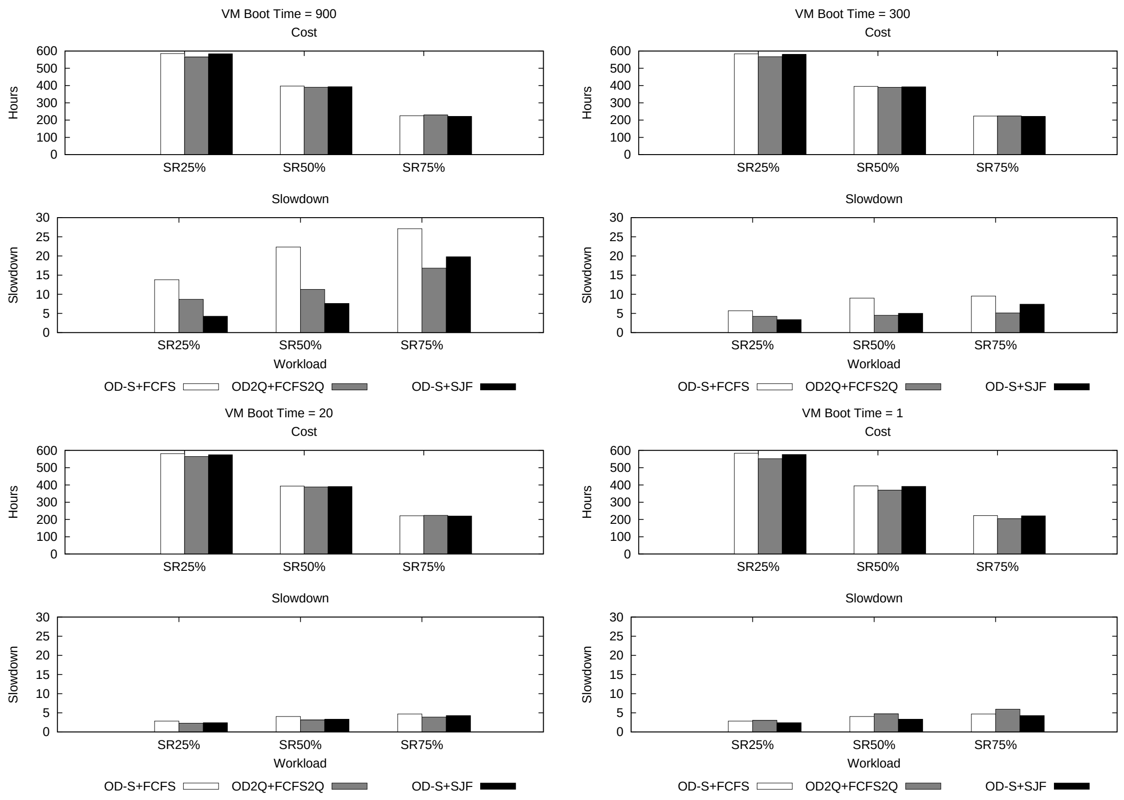


Figure 6.8: Effects of VM boot time

achieve a lower slowdown. Thus, we can ask whether this tradeoff is still desirable when VM provisioning time decreases.

Therefore, here we reproduce the experiments from section 6.5.3, varying VM booting times and accounting for the observed changes in metrics such as slowdown and cost. For each of these experiments, we modify the *VM_BOOT_TIME* parameter from the simulator, originally set to our empirically oberserved value of 600 seconds, to the values of 900, 300, 20 and 1 second. Figure 6.8 shows the outcome, while the results for the original parameter of 600 can be seen in figure 6.6.

From this experiment, we can appreciate that, for this type of workloads, cost is mostly unaffected by the VM's booting time. However, there is an important difference in the average job slowdown, since most of the time jobs spend in waiting is due to VM startup. The first conclusion that we can extract is that differences between policies have the highest impact when booting time is high. In effect, for the case of instantaneous VM booting —corresponding to bottom-right graph in the figure—, variations in slowdown between the different policies is minimal. The second insight from this experiment is related to the benefits of the multi-queue provisioning and allocation policy pair. High booting times, as depicted in the top-left corner, affect such policies negatively. In fact, for this scenario the `SJF` policy performs better for the `SR25%` and `SR50%` workloads. This can be attributed to the penalization of having VMs separated in two queues, with the result that jobs from one queue cannot reuse VMs from the other one. This is specially negative when the waiting time for new VMs is high, as in this case. On the other side, we can see how the benefits of the two-queue policies dwindle as the penalization for *thrashing* diminishes. In the case of instantaneous VM booting, all the achieved benefits from these policies are lost, and only the fragmentation issue remains, converting this pair of policies in the less beneficial choice.

6.6 Simulator Validation

The use of simulation throughout this chapter fulfills an important role to support the empirical experiments executed in actual IaaS providers. The cost associated to running such experiments, in terms of time, energy and economic requirements, greatly restricts the scope of the tests. Conversely, the data acquired from such executions is specially valuable since it captures all the underlying details from each platform, such as the effects of concurrent VM provisioning on disk and network, the performance associated to each virtual manager implementation, or the effects of the virtual machine monitor, to name a few.

Even though our simulator tries to replicate the behavior of Skymark —and in fact, it shares implementation details such as the submission or provisioning and allocation policies' code—, it is necessary to establish a baseline to ensure the results produced with it are close to those achieved by the real execution experiments. In this section, we define such a baseline by replicating some of the experiments in previous sections to give a minimum set of guarantees in terms of the simulator's expected behavior.

For all the experiments in this sections, we reproduce the runs from the FIU site to compare the effect of different provisioning policies for different types of workloads, Uniform, Increasing and Bursty, as defined in section 6.4.3. We create synthetic traces with the same properties as the ones employed in the real testbeds, with the difference that instead of actually executing computational jobs on the virtual resources, we replicate their behavior in terms of arrival distribution and duration. This fact rules out the comparison with the *ExecKN* policy, which uses specific job parameters to estimate their runtimes.

In oder to establish a baseline, we perform repeated runs for each of the experiments, and compare two of the fundamentally defining values for each of the policies, namely *slowdown* and *execution cost*. The rest of the attributes used in this chapter can be derived from these. Then, we calculate the resulting mean across executions and the mean's confidence interval. By placing the results obtained from real testbeds among the obtained range, we can assert that the simulation is sufficiently accurate for our purposes.

For each of the validation experiments, we generate workloads following the original model in terms of overall duration, job length average and standard deviation, arrival distribution and system load. Each workload is executed 30 times, and a 95% confidence interval is calculated for the resulting mean.

Figures 6.9, 6.10 and 6.11 depict the results of the experiments. As it can be observed, our simulator provides close results to the real environments in terms

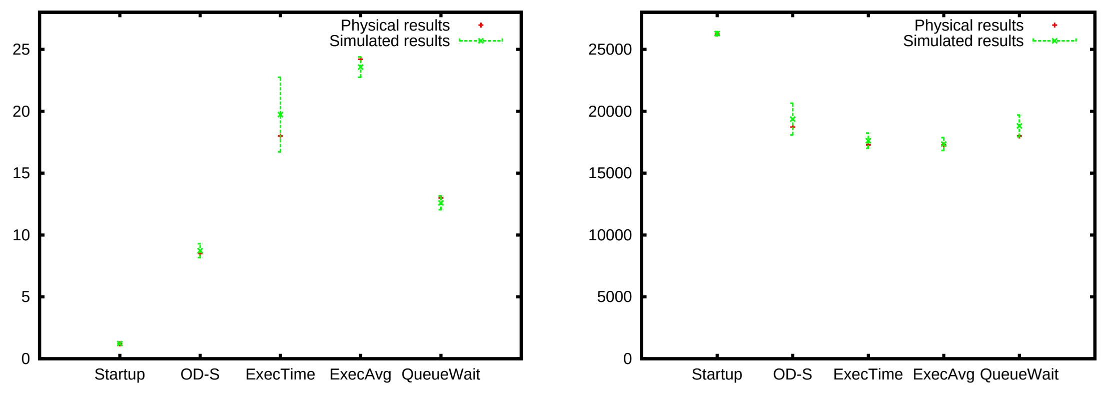


Figure 6.9: Simulator validation for steady workload

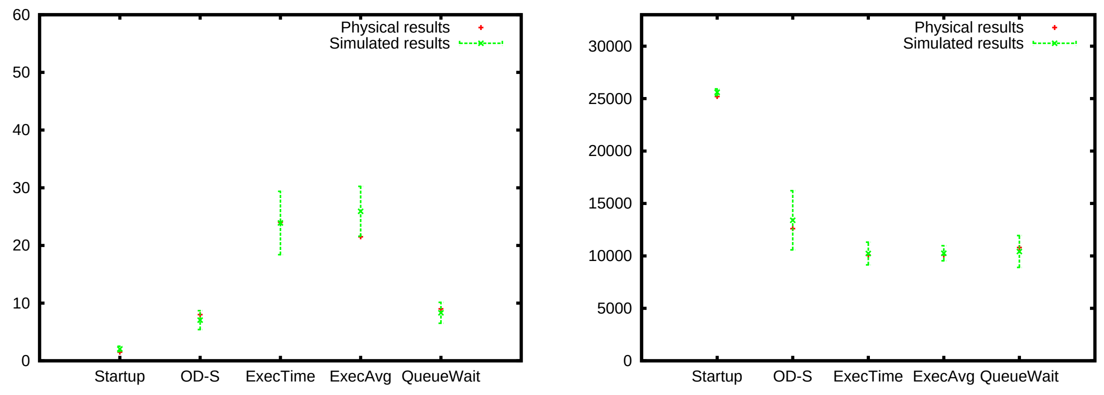


Figure 6.10: Simulator validation for increasing workload

of slowdown and cost. Additionally, we can extract additional insights from the experiments regarding the variability of the simulated results. First, the `Startup` policy tends to result in very close slowdown and cost values, confirming that the most important component in these metrics results from VM provisioning decisions. Second, we can see that the `ExecTime` policy is highly sensitive to variations in the generated workloads, since it makes decisions based on the actual execution time of the jobs. Contrarily, the `ExecAvg` policy has a lower variation for the uniform workload since it averages job durations to determine when to provision new VMs. This effect does not hold for the increasing and bursty workloads, since the averaging of past executions cannot predict load increases.

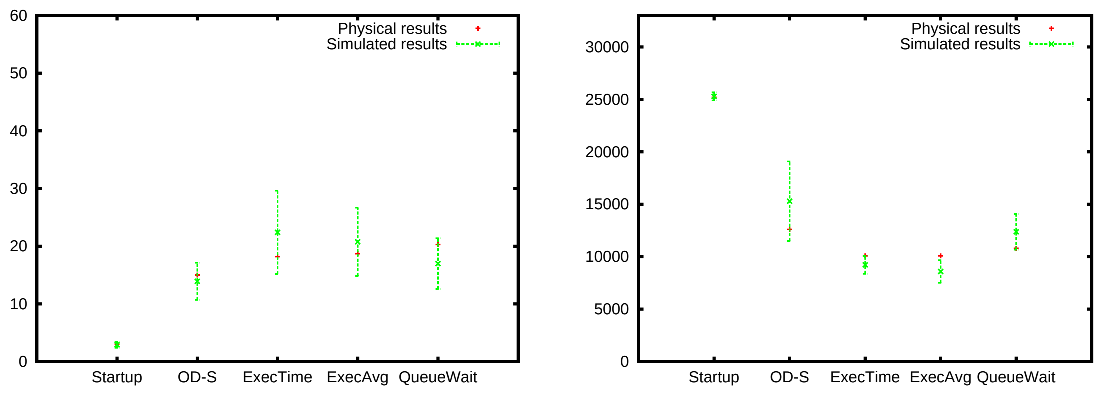


Figure 6.11: Simulator validation for bursty workload

6.7 Conclusion and Future Work

To manage their workloads, current and near-future users of IaaS clouds need a better understanding of the performance and cost of provisioning and allocation policies. In this work we have conducted a comprehensive study of these two types of policies, and of the interaction between them.

Overall, we have investigated eight provisioning and four allocation policies, and their interplay. We have developed SkyMark, a framework for IaaS performance evaluation, and conducted with it empirical research in three IaaS clouds, including Amazon EC2. Due to actual resource and budget availability constraints, we have also duplicated SkyMark functionality in a discrete event simulator. Based on results obtained in real and simulated IaaS clouds, we conclude that none of the tested (combined) policies is consistently better than the others across all cases. Our five main findings are summarized below:

1. `OD-ExecTime` and its two variants (especially `OD-ExecKN`), are a good performance-cost trade-off among the investigated provisioning policies;
2. Geometric on-demand provisioning policies, such as `OD-G`, are worse than single-VM on-demand policies;

3. The combined `OD-2Q–FCFS-2Q` policy, which we are the first to investigate in the context of clouds, is a good slowdown-cost trade-off for workloads with a significant ratio of short to long jobs;
4. Naïve static provisioning policies deliver stable performance, but incur up to 5 times higher cost;
5. Allocation policies with information about job runtimes achieve significantly better performance than uninformed allocation policies, when coupled with dynamic provisioning policies.

In the future, we plan to extend this work to consider new provisioning and allocation policies that adapt to changing workload, evolving resources, and complex Service Level Agreements. We will also investigate more diverse types of workloads, such as the typical workloads of grids [IE11].

CHAPTER 7

SCHEDULING PARALLEL WORKLOADS ACROSS VIRTUAL PROVIDERS

In the previous chapter, we discussed the relationship between the processes of virtual machine provisioning and job-to-VM allocation for single-processor tasks. Here, we take a step further and expand on our findings in two directions. First, a more general view of computational jobs is taken, in particular, trying to model multi-processor scientific applications. Such jobs show a degree of parallelism and different communication requirements: from tight job coupling with low latency needs to highly distributed or "embarrassingly parallel" tasks. We explore the effects of different provisioning and allocation policies when multiple processors —or VMs— are required, and propose improvements to strict queuing strategies such as First Come First Served, and relaxed ones such as First Come Best Fit. Secondly, we envision a novel brokering approach spawning from our work in inter-site meta-scheduling in chapter 4 where virtual resources from partnering sites are shared to execute scientific workloads. In particular, we explore how site provisioning and allocation policies, which as shown in the previous chapter can be controlled by users, can be considered in the site-selection stage of the broker to improve global metrics such as response time or cost.

7.1 Background

Most of the work in meta-scheduling considers the scenario of collaborating sites with locally owned physical infrastructure, usually exposed to an organization through a batch scheduler and shared to external partners through a public API or protocol. This model was introduced in chapter 4, where we defined and implemented an interoperable and de-centralized meta-scheduler.

However, the rapid emergence of virtualization represents an important change in how scientific workloads are executed. There are still applications that need close access to the hardware and high levels of customization to extract the most from physical resources, but a growing number of users are targeting virtual machines due to better control, higher flexibility and ease of use; thanks to the possibility of encapsulating all the necessary details to execute the application. Birkenheuer et al. [BBK$^+$12] discuss many of the benefits of using virtualization for High Performance Computing users, and describe new solutions to improve I/O and networking bottlenecks, which constitute the main problem for communication-bound software [LHAP06, PCI].

Hardware virtualization, and in particular Virtual Machines, provide new opportunities and challenges to HPC users. The most relevant characteristic related to scheduling is that resources can be acquired —and released— on demand, thus creating the possibility of energy savings to providers by multiplexing their hardware (consolidation) and turning off unused infrastructure. Furthermore, job submitters have a much larger pool of execution candidates, since all the required software stack —Operating System, libraries, executables and even input data— is self-contained in the VM image. Thus, assuming that members of a federation of providers employ compatible virtualization technologies, job requests can be off-loaded with more flexibility. Contrarily, users who executed specialized workloads in the grid needed out-of-band communication with system administrators to install required libraries or special compilation jobs to create the necessary binaries for each system, based on the available compilers, libraries or Operating System versions.

There are still limitations to inter-site execution of workloads, especially for applications that require very high volumes of data transfer, either as an input or as an output. However, cheaper storage and faster interconnections among partners is enabling more applications to be executed across organizational bounds. In this chapter we assume applications don't have high data dependency, either because collaborating partners already have the necessary input, or because these applications don't incur in high costs to transfer the required information in and out. Examples of the first case are collaborators in High Particle Physics such as CERN or Fermilab, where multiple sites (organized as tiers) may replicate detector data, or BioInformatics teams that cooperate in the study of a particular organism, thus hosting the required database across multiple sites. An instance of the second application type is weather simulation with WRF [MDG+04], where input and output data is relatively small.

In the previous chapter we explored some of the benefits of using virtualized resources and proposed new policies to take advantage of the fast provisioning of new infrastructure based on user demand. One of the most important conclusions consists in the benefits of on-demand provisioning in terms of cost savings versus traditional, always-on resources (exemplified by the *Startup* policy). In this chapter we extend our findings in three main directions:

1. We implement a more general model of scientific computation, considering parallel jobs. Such applications represent a high portion of existing workloads in clusters and grids [IJSE07]. Parallel jobs need to run in multiple resources concurrently for a period of time to produce the requested output.
2. We consider the effects that different provisionig and allocation policies have in the site-selection phase of meta-scheduling. In particular, we take advantage of the additional information each partner has about provisioning capabilities for other peers. With this additional information, we show that better forwarding strategies can be implemented that reduce VM thrashing (*i.e.,* overhead due to startup and shut-down) and job slowdown.

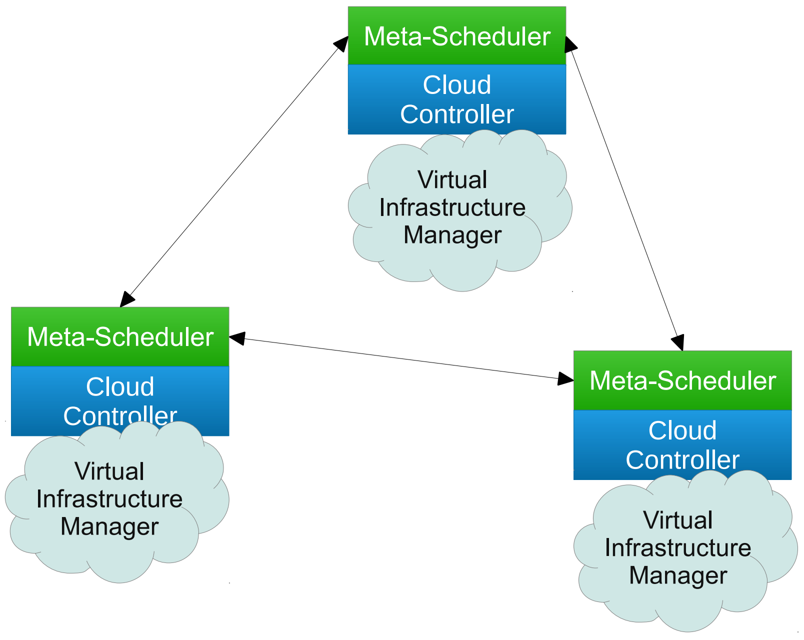

Meta-Scheduler
Cloud Controller
Virtual Infrastructure Manager
Meta-Scheduler
Cloud Controller
Virtual Infrastructure Manager
Meta-Scheduler
Cloud Controller
Virtual Infrastructure Manager

Figure 7.1: Virtual provider federation model

3. We use realistic workloads to better represent real submission patterns from HPC users. In addition to the synthetic executions used in the previous chapter, we implement a simulator capable of running thousands of parallel jobs across collaborating providers.

We propose a federation model where meta-schedulers are used to share virtual resources among peers, while exploiting the benefits provided by virtual infrastructure, in terms of flexibility and additional control over how and when resources are created. Figure 7.1 shows our general approach combining a meta-scheduler component and a cloud controller. While the first one manages communication with other members of the federation, the second is, in charge of provisioning VMs and allocating jobs to them.

The rest of the chapter is organized as follows: In section 7.2, we give an explanation of the main ideas used through the chapter, including the design of our simulator, the employed traces and the virtual resource brokering model. In section

7.3 we discuss the different policies considered. Section 7.4 contains experimental results and a discussion of their significance. Finally, in section 7.5 we validate the used simulator to provide confidence intervals for the experiment results.

7.2 System Model

The work in this chapter combines and expands on two of our already discussed contributions. In the first place, we take the peer-to-peer distributed meta-scheduling model discussed in chapter 4 and extend it to consider virtual providers. Secondly, we build upon the simulation capabilities of Skymark, already described in chapter 6, to be able to measure long execution traces across different providers. While the objective of the previous chapter was to give a realistic view of job scheduling in actual virtual providers such as OpenNebula or Amazon EC2, here we want to take a broader perspective in order to study the effects of meta-scheduling, provisioning and allocation policies on large workloads and deployments, something that is difficult to achieve using real-time execution in existing providers due to high costs and very long experiments, in the order of days or weeks.

7.2.1 Brokering Across Virtual Providers

In our peer-to-peer meta-scheduling architecture, described in section 4.2, we introduced a communication protocol to enable the offloading of jobs among collaborating sites with physical resources. The protocol is divided in three main aspects: communication, resource information and job execution. Our proposed model assumes that each site has a collection of physical resources, which can be heterogeneous, that are controlled by a Local Resource Management System (LRMS) and made accessible to other partners through a concrete implementation of the protocol. In our particular instance of the FIU meta-scheduler, a cluster running Sun Grid Engine [Gen01] and the Globus middleware [FK96] were joined to external sites through the protocol.

The main role of the meta-scheduler is to determine the most appropriate peer to run a particular job. This process is known as site-selection. In our system, users send jobs described by the JSDL [ABD$^+$05] format, which allows the specification of binaries, hardware and software requirements and the desired degree of parallelism. The job execution interface of the meta-scheduler receives the submissions and determines whether to run the jobs on local resources, or to forward the jobs to other sites, following a site-selection strategy. Such strategies may consider different information provided through the meta-scheduling protocols, such as the number of resources available at other locations, or locally computed data, such as the number of jobs successfully run in the past.

However, one of the requirements for this approach is that each site maintains independence from its partners. LRMSs at each site are controlled by local administrators, and therefore no assumptions can be made by a meta-scheduler about the scheduling policies used internally. Additionally, resources are fixed at each site (with the exception of failures or upgrades, which are not the norm), and therefore the job of the meta-scheduler is rather limited.

With the advent of virtualization, this scenario changes considerably. Virtual providers are in charge of instantiating Virtual Machines per request of users, and while there is no external control over how VMs are allocated to physical resources, users can control how many VMs are provisioned, and how jobs are mapped to them. This idea is similar to Virtual Clusters [CIG$^+$03, RMNC06, MKFG09], although in our case provisioned VMs don't have a long-term life unless there is demand for them. Our scenario is more dynamic than Virtual Clusters, focusing on the better usage of resources than on middleware installation, monitoring or user management.

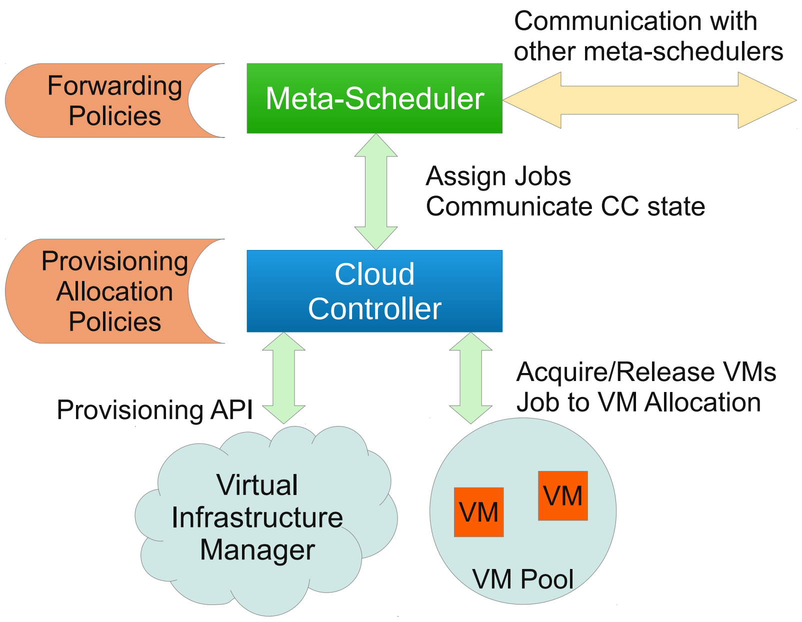


Figure 7.2: Virtual provider model

In our model, we divide each partner in the federation in three main components, as shown in figure 7.2: The Meta-scheduler, the Cloud Controller, and the Virtual Infrastructure Manager. The Meta-Scheduler is in charge of communication with other peers to share resource information and off-load jobs between sites according to a particular forwarding policy. The Cloud Controller has a similar role to Skymark, and is in charge of provisioning new VMs and allocating jobs to them, according to the defined provisioning and allocation policies. Finally, the Virtual Infrastructure Manager has the capabilities of instantiating VMs on the physical substrate and managing VM images. The Cloud Controller interacts with the Virtual Infrastructure Manager through an available API. Examples of Virtual Infrastructure Managers are OpenNebula, Eucalyptus or Amazon EC2.

The Cloud Controller shares information about the employed policies, the current state of the resources, and the available capabilities. The dynamic nature of virtual infrastructure can directly impact site-selection strategies by making much more

Physical Providers	Virtual Providers
Number of available machines	Maximum number of VMs
Queue length	Available VMs
Average job waiting time	Provisioning policy
Successful job ratio	Allocation policy
	VM uptime
	VM idle time

Table 7.1: Comparison of physical and virtual provider information.

information available to the meta-scheduler. Table 7.1 lists some differences in the available information that the meta-scheduler can use to perform site-selection.

7.2.2 Simulator

In order to study the effects of different policies on workload execution across sites, we extended our work in the simulation embedded in Skymark, presented in last chapter, to process more complex workloads and multiple sites. The simulator processes events generated from user job submissions and computes the steps at a virtual provider from the job arrival time until it is assigned to a VM and finalized. Provisioning and allocation policies can be defined to control how VMs are provisioned and how jobs are allocated to them.

The first change added to the simulator is to account for multiple sites, each controlled by a meta-scheduler. In this new model, multiple collaborating organizations can be defined in the simulator. Each organization publishes internal functionality through a meta-scheduler and is able to receive jobs from users and also from other meta-schedulers. The meta-scheduler receives a job, signaled by the `JOB_ARRIVAL` event, and determines where to assign it, based on the information from other peers and the employed *forwarding policy*. Then, it generates a `JOB_ASSIGN` event to transfer the job to a particular provider, which adds it to a local queue. Virtual Providers are in charge of running jobs on their own resources by provisioning new VMs (or reusing existing ones) and allocating the queued jobs to the VMs according to an allocation policy. Meta-schedulers may decide to re-submit already assigned jobs to a more appropriate site based on internal metrics such as the time the job waited in the queue or updates in peers' information, such as load change.

The second addition to the simulator is implementing the capability to run parallel jobs. In our previous contribution we consider only single processor workloads, but in order to study a wider range of scientific traces we examine the effects of multi-processor tasks. For this, we added a new attribute to jobs, where required CPUs can be specified. The addition of multi-processor jobs can lead to starvation depending on the employed policies, and different optimizations need to be made to improve throughput and reduce slowdown when they are present.

While there are virtual providers that offer multi-CPU VMs, we opted to assign one VM per requested processor. For example, a 16 cpu jobs would be assigned to 16 VMs that would run in parallel. Our assumption can be justified by the fact that running each task in a different VM gives a better isolation among processes and reduces the possible contention originated by running multiple jobs in one single machine.

7.2.3 Workloads

For this work, we also aim to explore the behavior of the different policies on large workloads and sites in order to have a better idea of the tradeoffs caused by the choice of strategies. We use real scientific traces to replicate realistic scenarios in the field of HPC. We focus on two of them, each representing two different types of jobs. In the first case, we run a fragment of the Cornell Theory Center SP2 trace, from the Parallel Workloads Archive [Fei08], representing a week of execution on

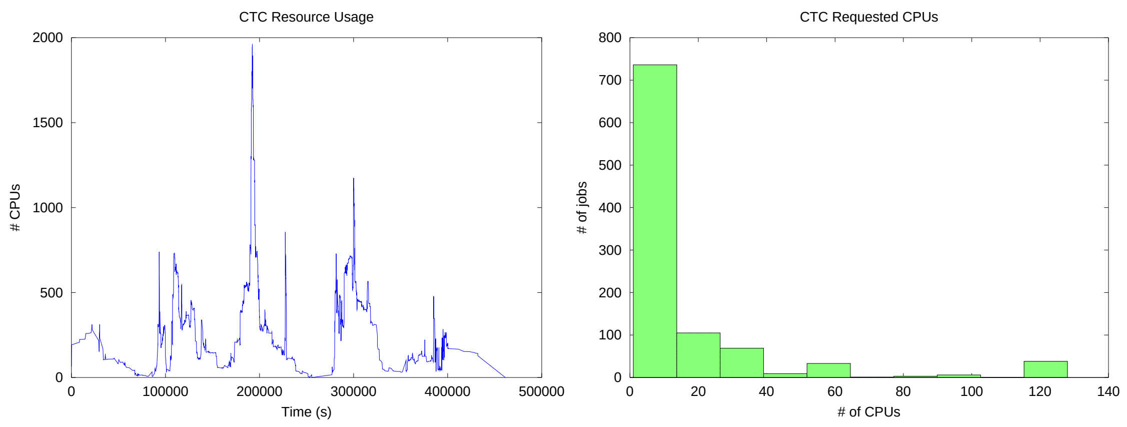


Figure 7.3: CTC Trace characterization

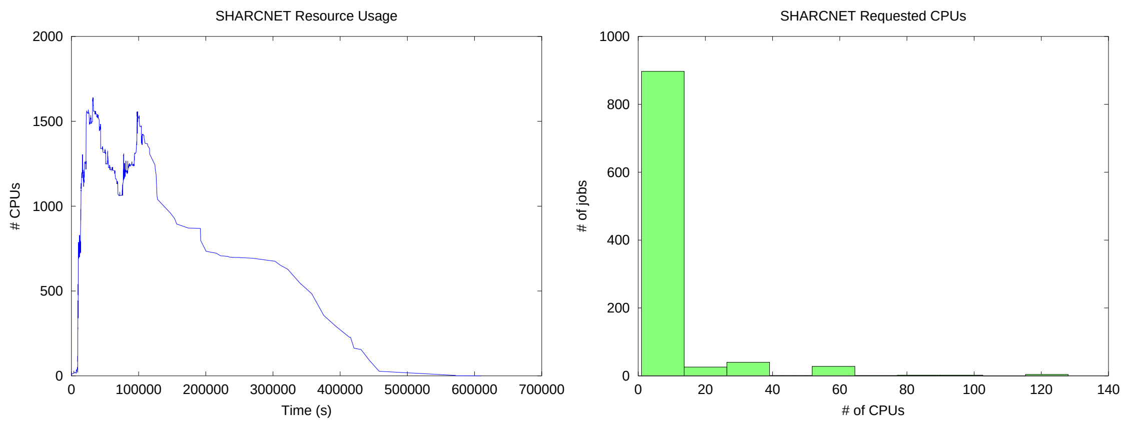


Figure 7.4: SHARCNET Trace characterization

approximately 450 machines. Secondly, we use the SHARCNET trace from the Grid Workloads Archive [ILJ+08], which consists of a grid workload. Figures 7.3 and 7.4 show two aspects of each workload. On the left, the graphs illustrate each workload's ideal queue length, assuming there are infinite resources. The right figures contain the traces' cpu request distribution.

As it can be seen in the figures, these two traces represent very different HPC paradigms. The CTC trace represents a workload sent to a dedicated cluster with highly parallel, tightly coupled applications such as WRF. As the left graph shows, jobs arrivals are evenly distributed through time, following day-night submission patterns. The histogram to the right depicts that, even though most jobs are single-processor, there are many multi-processor requests. In the second case, the SHARC-NET workload evidences the use of Bags of Tasks in grids [IJSE07], this is, usually best effort, loosely coupled jobs that can be easily distributed among partnering organizations. The queue lenght graph on the left shows an uneven arrival pattern with a higher concentration of arrivals at the beginning and a cool-down period (showed by a long tail) at the end. The cpu distribution graph on the right points to a majority of single-processor jobs. On average, the second trace has an ideal utilization of approximately 1100 cpus with an infinite number of resources.

7.3 Policies

In this section we describe the different studied policies. We first discuss the differences between allocation policies that execute jobs in strict order of arrival and those that can move jobs from the back of the queue. An important consideration here is that we are assuming the user doesn't have exact information about the job's runtime. Mu'alem *et al.* [MF01] studied the difficulty that users face when asked to give good estimates of their jobs' running time. This constraint rules out the `BACKFILL` policy, which we do not consider here.

7.3.1 FCFS vs. FCBF

There have already been studies comparing scheduling policies that consider strict order of arrival (concretely First Come First Serve or `FCFS`) and others that try to optimize resource utilization by moving jobs forward in the queue when they are better suited to run than the one at the head. We call this family of policies First Come Best Fit (`FCBF`), since there is still a concept of ordering, but more appropriate jobs are favored. In `FCFS`, a job requesting multiple processors can block other jobs if there are not enough available resources to service it, even though other jobs could be moved forward in the queue to utilize them. However, in the case of `FCBF`, an aggressive policy promotes small jobs to decrease the average waiting time, at the cost of potentially causing starvation for jobs with large resource requests.

The reason we need to revisit these policies in this work is that existing research considers physical —and therefore static— resources, while in our model, provisioning policies may favor on-demand instantiation of VMs. These policies, as discussed before, bring important benefits in execution cost. However, they change how `FCFS` and `FCBF` behave. For example, in the case of partially available resources to serve a job, the allocation policy has three possibilities: To wait until there are available VMs, to provision new ones, or to move jobs that can run ahead in the queue. One important consideration is that partially available resources need to be marked until there are enough to fulfill the number of required processors by a job, so that they are not released due to being idle. We implemented this feature by allowing allocation policies to *reserve* VMs, which prevents provisioning policies to stop them.

Based on the enumerated possible decisions, we have implemented a dynamic `FCFS` policy and three different variants of `FCBF`. `FCFS` always reserves any free VMs until there are enough to run the job at the head of the queue. For the `FCBF` family of policies, we implement three types. The fist one, `FCBF-Aggressive`, always promotes jobs to the front of the queue if there are enough VMs to run any of them. Otherwise, VMs are reserved for the first one conditionally, but this condition is revisited at every scheduling step. The second variant, `FCBF-Split`, divides the available VMs for the job at the head and for other jobs (assigning the remaining ones at the end of the process to the head). Finally `FCBF-QWait` promotes jobs in the back of the queue unless the job at the head accumulates a waiting time higher than the specified threshold.

7.3.2 Site Selection

The second group of strategies we focus on in this chapter is forwarding, or site-selection policies. As explained before, while provisioning and allocation policies reside in the Cloud Controller component, the meta-scheduler, which is in charge of receiving user requests and negotiating execution among peers, may implement different strategies for off-loading jobs. While chapter 4 explored some of these strategies for physical providers, here we are interested in evaluating their role when peers are virtual providers.

As a baseline, we implemented a basic strategy, `RoundRobin`, which alternates the selected destination site among all the peers plus the local one. `RoundRobin` results in a fair usage of sites, when they have approximately the same size, although differences in load or provisioning capabilities require more advanced strategies.

A second groups of policies takes capacity into account, in order to select the appropriate site when they have different sizes. First,we implemented `HighestCap`, which chooses the site with the highest capacity, this is, potential number of VMs that can be used. For example, a site that can instantiate a maximum of 100 VMs and has 50 of the in use and 20 idle, it would have a capacity of 50 VMs (20 idle plus 30 potential). The second policy in this group, `HighestIdle`, forwards jobs to the site with the highest number of idle VMs. `BFCap` assigns jobs to the site with the closest capacity to the number of requested processors by the job. Finally, `BFIdle` selects the site with the closest number of idle VMs to the size of the job.

7.4 Experimental Evaluation

In this section we simulate the submission, scheduling and execution of scientific workloads to study the behavior of the policies described in the previous section. First, we compare the use of `FCFS` and `FCBF` policies for a single site. Then, we study different forwarding policies.

7.4.1 Experiment Setup

For the following experiments, unless noted otherwise, we set a scheduling interval of 20 seconds at the meta-scheduler and 10 seconds at the cloud controller. VMs are considered homogeneous and single-processor. VM booting time is 600 seconds, and shutdown time is 60 seconds.

In order to evaluate each experiment, we use the following metrics:

- *Makespan*: the overall time from arrival of the first job until the last job is finished.
- *Slowdown*: the average factor by which each job is slower than it would be if there were enough resources for it to execute.
- *Bounded slowdown*: similar as slowdown, but only considering jobs with a duration over a certain number. We set this threshold to 10. The duration of jobs under this threshold is set to the threshold.
- *Average Weighted Response Time (AWRT)*: the ratio of job waiting time, weighted by the job duration and requested CPUs.

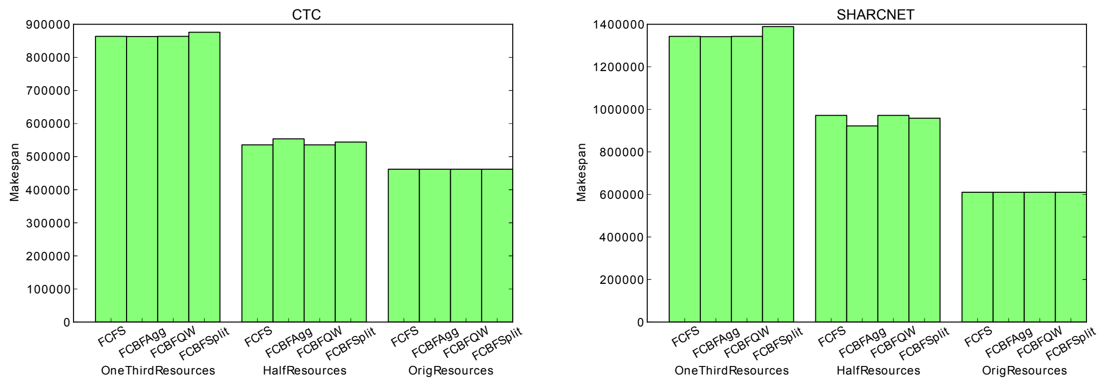


Figure 7.5: Makespan for `OnDemand` plus `FCFS` and `FCBF` policies

7.4.2 Single-Site Parallel Execution

First, we want to explore the different trade-offs of allocation policies with strict order-of-arrival job servicing versus those that allow promotion of jobs in the queue to the first position based on different heuristics. The initial experiment processes the CTC workload at one site, varying the number of maximum VMs. For each trace, we consider the cases of one third of the original resources, half of the resources, and the same number of resources as in the original trace. By varying the number of resources, we can explore the behavior of the policies under different contention levels.

We compare the results for the `OnDemand` provisioning policy and the allocation policies introduced in section 7.3.1: `FCFS`, `FCBF-Aggr`, `FCBF-QW` and `FCBF-Split`. We set a threshold of 300 seconds for `FCBF-QW`. Figure 7.5 shows that the workload's makespan is mostly determined by the number of resources at the site. Since the workloads don't have a continuous usage of the infrastructure, the overall execution time is the same for both traces. However, the impact of the policies can be appreciated for individual jobs, as showed next.

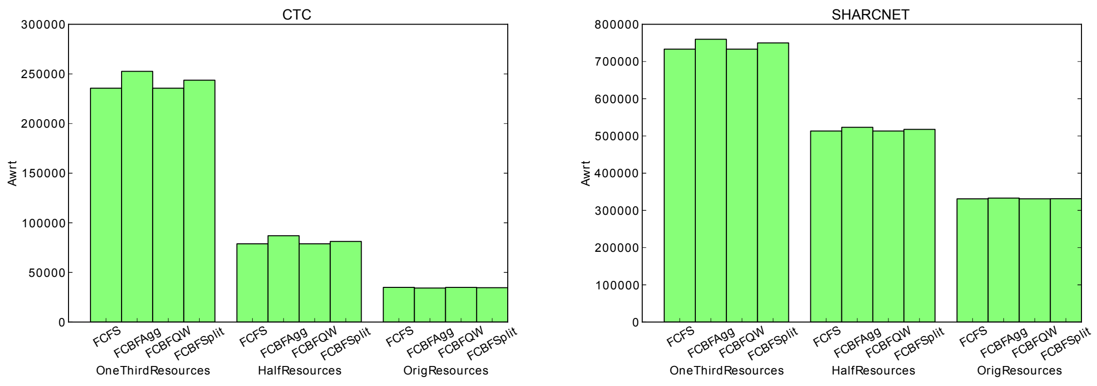


Figure 7.6: AWRT for `OnDemand` plus `FCFS` and `FCBF` policies

Figure 7.6 depicts the AWRT metric, which gives a factor that illustrates the waiting time, weighted by each job's size in terms of running time and number of requested processors. Here we can see that the `FCBF-Aggr` policy does have a bias against jobs requesting multiple processors. This can be specially appreciated when there is a high resource competition. The `FCBF-QW` policy achieves the same AWRT as FCFS, which is the one with the fairest behavior.

Finally, figure 7.7 shows the bounded slowdown for each of the tested policies. This experiment reveals how aggressive `FCBF` highly reduces the average job slowdown. Overall, jobs have to wait up to five times less to execute when this policy is used, at the expense of having jobs that request many processors wait longer. However, since the number of jobs with few processors is very high in the studied traces, this is a reasonable choice for the workloads studied here.

In conclusion, this experiments highlights how for parallel grid workloads, aggressively promoting jobs to the front of the queue leads to better resource utilization, with a small cost for large jobs. There is still the possibility of starvation for large jobs; however, overall resource load should be much higher and constant for this to happen. Based on the traces explored in this work, the benefits outweight the costs.

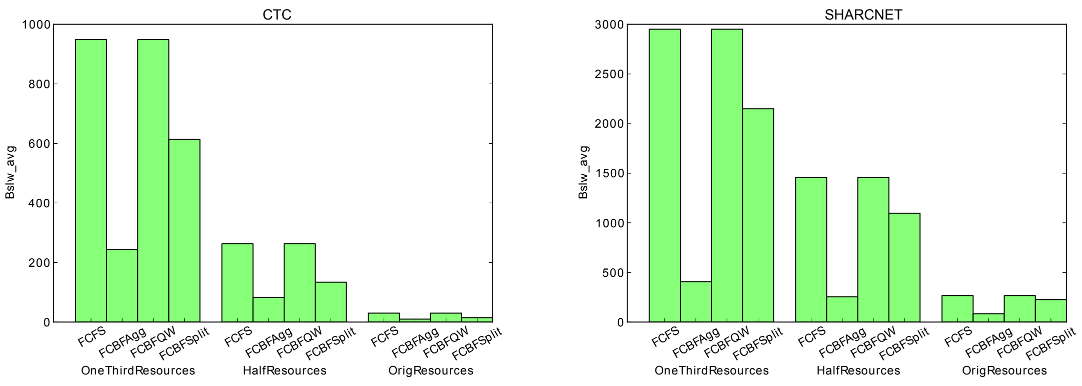


Figure 7.7: Bounded slowdown for `OnDemand` plus `FCFS` and `FCBF` policies

7.4.3 Forwarding policies

The second set of experiments compares the effects of forwarding policies at the meta-scheduler level when there are multiple sites that can instantiate virtual resources. As explained at the beginning of the chapter, virtual providers can offer richer information to be used by the meta-scheduler in the process of site-selection. In this experiment we compare the execution of the CTC and SHARCNET traces in one site with the maximum resources —450 for the first, 1100 for the second—, and in two sites with different capabilities. We assign one third of the resources to one site and two thirds to the other to showcase the effects of unbalanced behavior.

Then, we measure the effectiveness of different forwarding policies to select the most appropriate site to run jobs. Considering the `OnDemand` provisioning policy to instantiate VMs when more resources are required, we explore the behavior of two groups of policies. The first selects sites based on the maximum number of potential VMs, or the site's capabilities; the second sends jobs to the site with higher idle VMs. For each of these two types of policies, we create a variant for highest number of resources and for closest number of resources. Thus, we evaluate the `HighestCap`, `BFCap`, `HighestIdle` and `BFIdle`. We also consider `RoundRobin` as a baseline.

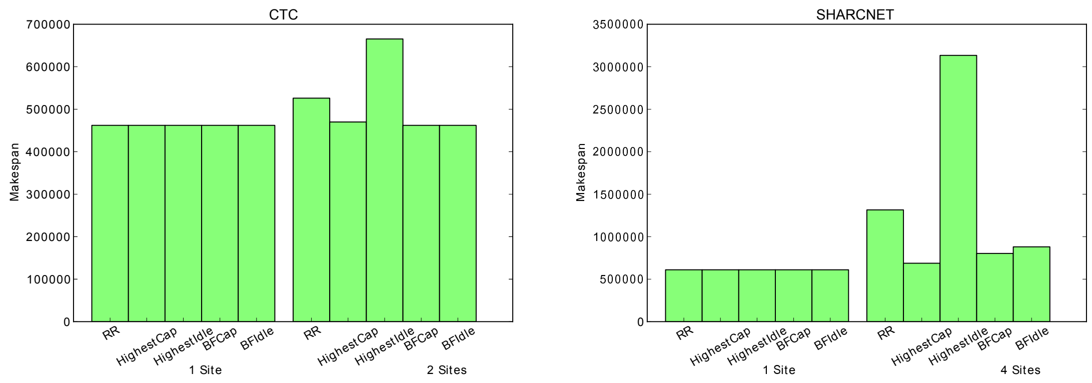
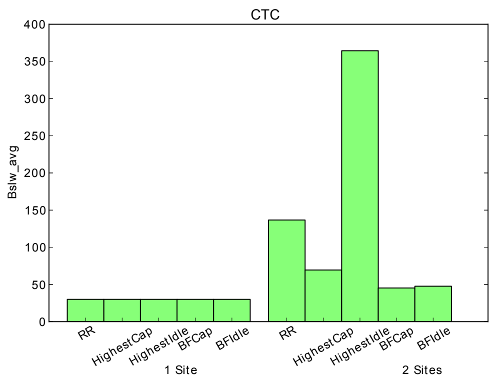


Figure 7.8: Makespan for forwarding policies in heterogeneous sites

Figure 7.8 shows that, besides the `RoundRobin` and `HighestIdle` policies, the overall makespan is similar for the CTC and SHARCNET traces when resources are concentrated in one site or distributed among multiple sites. The bad behavior of the mentioned policies can be explained by the differences in site capabilities for `RoundRobin`, which blindly alternates among peers. In the case of `HighestIdle`, the policy favors that site with most available VMs, without accounting for the actual capacity. This effect doesn't happen with `BestIdle`, since the latter strategy chooses more conservatively the target execution site.

Figure 7.9 depicts the bounded slowdown metric, which confirms the trend exposed in the previous chart. Here, another difference is that `RoundRobin` has better results for the SHARCNET trace, because both sites have a higher number of resources and there is never as much contention in one location as in the CTC experiment. In general, however, we can conclude that for virtual providers with `OnDemand` provisioning policies, considering the *potential* capabilities provides better site-selection decisions than just looking at the available resources at a given point in time.



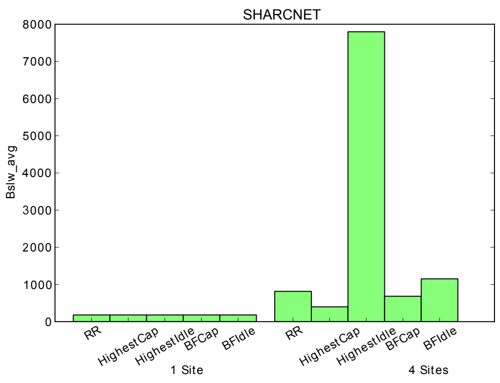


Figure 7.9: Bounded slowdown for forwarding policies in heterogeneous sites

7.5 Simulator Validation

As we did in section 6.6, here we try to narrow the accuracy of our simulator in order to provide quantitave confidence ranges to situate our results. As previously discussed, the simulator is an evolution of the one used in conjunction with Skymark in the previous chapter, adding support for inter-site job execution through forwarding policies.

Therefore, we attempt to provide a baseline to validates the obtained results by measuring the simulator's behavior for the remaining functionality. Since the execution of jobs inside of a site through the use of allocation and provisioning policies was already studied and validated in chapter 6, the remaining work consists in doing the same for forwarding policies.

Since the contents of this chapter are a natural extension of the grid metascheduler architecture, it makes sense to consider whether our simulation can faithfully reproduce the results from chapter 4. Clearly, there is an important difference between the two approaches, since the present chapter considers virtualization while the former only performs job execution on physical resources. However, our simulator already implements the `Startup` provisioning policy, which aims to replicate the behavior of physical sites by starting all VMs before the jobs are sent to the system.

We choose the experiment from section 4.5.2, which executes a part of the Cornell Theory Center workload using the unmodified meta-scheduler protocol and a real-time emulator in order to validate the simulator. For all validation experiments, we set the `Startup` provisioning policy as discussed before, then the `FCBF-Aggr` allocation policy to replicate the First-In First-Fit site scheduling from the emulator, and define the three site configurations discussed in chapter 4. We execute the experiments for the four different forwarding policies with each of the site configurations and measure the resulting bounded slowdown.

In order to provide variability for the experiments, we select multiple sections of the CTC trace with similar characteristics. Each of the workloads is processed using the same scripts as in the previous experiments and the number of CPUs is capped to 128. Then, we repeat the experiments for 10 workloads and calculate the mean and 95% confidence intervals. Finally, we plot the results obtained from the meta-scheduler and emulator in the graphics.

Figures 7.10, 7.11 and 7.12 show the simulated and empirical results for this experiment. As it can be noticed, the results obtained from the real meta-scheduler implementations and emulated physical resources fall within the ranges of confidence for the simulator. We have observed that the ranges of confidence are relatively wide, and this can be explained because of some degree of variability in the results. We attribute this variability to small differences between the sections of the workload chosen. Even though we choose periods that are close in time to minimize high differences, there is an unavoidable digression that can be assigned to human submission patterns. However, we can still see that the trends are well defined and agree with the already obtained results.

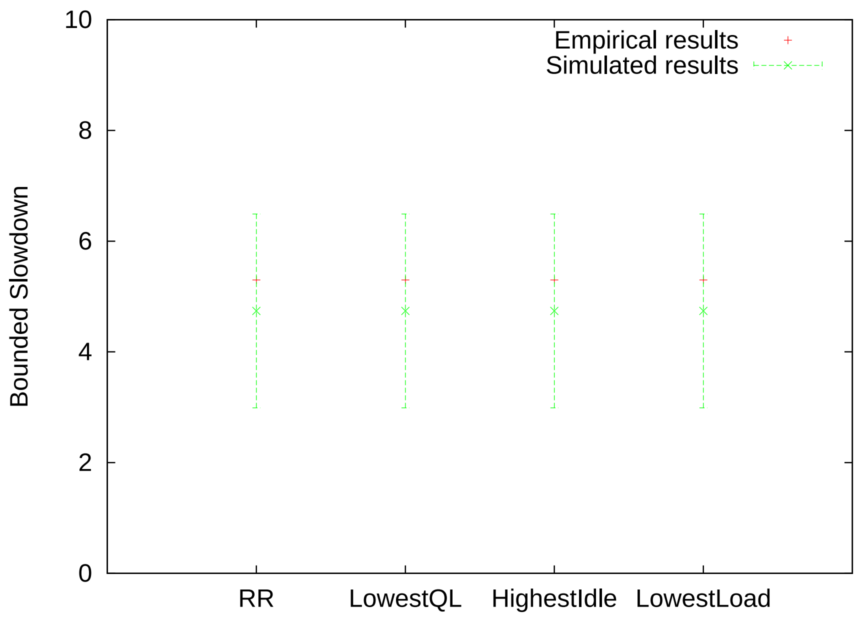


Figure 7.10: Simulator validation for 1 Site

7.6 Conclusion

In summary, in this chapter we have studied different effects when scheduling scientific parallel workloads between virtual providers with policy-based allocation, provisioning and forwarding. First, we explored the effects of allocation policies for parallel jobs, and in particular how aggressive scheduling with on-demand resource provisioning can in general reduce the average waiting time at the expense of a slightly higher slowdown of large jobs. Secondly, we compared different site-selection algorithms at the meta-scheduling level when sites can provision virtual infrastructure. Here we learned that policies that consider site capacity, or potential number of VMs, generally perform better than those who take into account dynamic load, or number of idle resources at a given time.

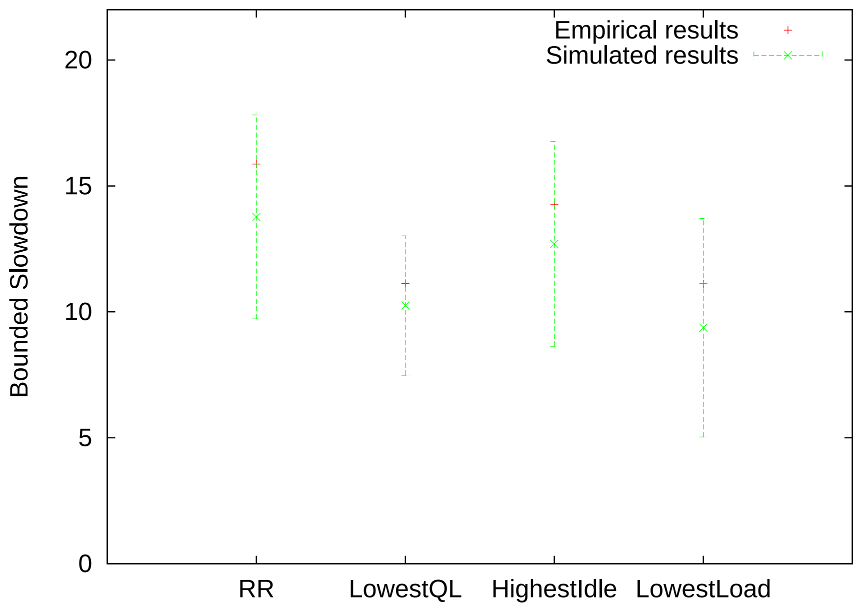


Figure 7.11: Simulator validation for 2 Sites

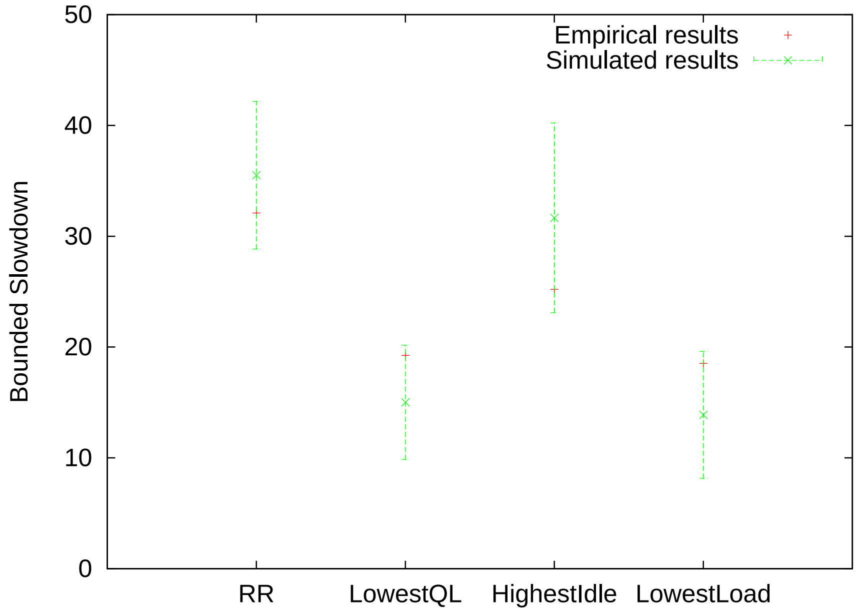


Figure 7.12: Simulator validation for 3 sites

CHAPTER 8

SUMMARY

In this dissertation we have identified the opportunities and current problems of site federation to execute computational jobs on virtual and physical infrastructure. Presently, a number of objectives have already been met by producing publications in high impact scientific conferences and journals. This demonstrates the feasibility and relevance of our proposed directions.

We have developed a grid meta-scheduler protocol capable of forwarding computational jobs between different institutions with heterogeneous grid middleware implementations [BFL$^+$08, LVB$^+$09]. The Gridway meta-scheduler [HML07] has been improved to enable coordinated sharing of resources and the system has been tested by defining a resource emulator capable of running the unmodified sharing protocol between three sites.

We have proposed an initial federation model [VBR$^+$12] by identifying three different layers where sharing can be performed. We have determined the general mechanisms required to either translate requests among layers or to exchange them between providers. We also motivated the model with the example of a scientific weather forecasting portal.

A study of allocation and provisioning policies for clouds has been conducted by identifying different strategies and testing their joint behavior in real environments, consisting of three different sites at FIU, TU Delft and Amazon EC2 [VASI12]. We have proposed new policies to perform coordinated job to resource mapping and resource provisioning that improve workload metrics over existing ones.

Finally, we have described a new model for meta-scheduling among virtual resource providers, extending our work in site brokering and HPC job execution over distributed domains. We have studied the effects of different site-selection policies for peers to select where to offload execess jobs.

8.1 Future Work

In the future, there are three main directions in which this work can be expanded. In the first place, the developed tools and protocols can be used to test new policies to provide improved execution of scientific jobs across domains. The decentralized meta-scheduling model makes it easy for new institutions to join their resources to a federation while maintaining complete control over how they are leased. This also applies to the field of cloud computing, and in particular IaaS providers, which could make their infrastructure available to scientists through our protocol. Examination of new workloads would bring new insights about better forwarding strategies.

Second, the multi-layered federation model can be implemented starting from our work in the meta-scheduler component. Besides the study of forwarding —or *federation*— policies, which direct the flow of jobs in the horizontal plane, good strategies for the *delegation* of requests from the higher to the lower layers have to be devised. This would enable the seamless execution of user applications in our system by translating high-level needs into job and VM requirements. Furthermore, the system would be extended to consider application-specific information into the forwarding strategies.

Finally, we envision a richer model of sharing virtual resources and running applications on them, where instead of exchanging “flat” VMs among federation partners, users could specify the application needs as a whole, including network requirements and the relationships between the provisioned VMs. This would allow an even richer set of federation strategies where parts of an application could be executed at different providers. Currently, this is not possible since parallel jobs are assumed to be homogeneous, but there is a large number of workloads outside of the realm of scientific computing that could benefit from this approach, for example Web applications.

[ABD$^+$05] A. Anjomshoaa, Fred Brisard, M. Drescher, Donal Fellows, et al. Job Submission Description Language (JSDL) Specification Version 1.0, GFD-R.056. Technical report, Open Grid Forum (OGF), November 2005.

[ACP95] T.E. Anderson, D.E. Culler, and D. Patterson. A case for now (networks of workstations). *Micro, IEEE*, 15(1):54 –64, feb 1995.

[AFF$^+$01] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, and B. Rochwerger. Oceano-sla based management of a computing utility. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 855 –868, 2001.

[AFG$^+$09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing, Feb 2009.

[ATN$^+$00] Kento Aida, Atsuko Takefusa, Hidemoto Nakada, Satoshi Matsuoka, Satoshi Sekiguchi, and Umpei Nagashima. Performance evaluation model for scheduling in global computing systems. *Int. J. High Perform. Comput. Appl.*, 14(3):268–279, August 2000.

[Azu] Microsoft azure. `http://www.microsoft.com/azure`.

[B$^+$07] Rosa Badia et al. *High Performance Computing and Grids in Action*, chapter Innovative Grid Technologies Applied to Bioinformatics and Hurricane Mitigation, pages 436–462. IOS Press, Amsterdam, 2007.

[BAGS02] Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002.

[BBA10] R. Buyya, A. Beloglazov, and J. Abawajy. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010)*, Las Vegas, USA, 2010.

[BBK$^+$12] Georg Birkenheuer, Andr Brinkmann, Jrgen Kaiser, Axel Keller, Matthias Keller, Christoph Kleineweber, Christoph Konersmann, Oliver Niehrster, Thorsten Schfer, Jens Simon, and Maximilian Wilhelm. Virtualized hpc: a contradiction in terms? *Software: Practice and Experience*, 42(4):485–500, 2012.

[BDF$^+$08] N. Bobroff, G. Dasgupta, L. Fong, Y. Liu, B. Viswanathan, F. Benedetti, and J. Wagner. A Distributed Job Scheduling and Flow Management System. *ACM Operating Systems Review*, 42:63–70, January 2008.

[BFL$^+$08] N. Bobroff, L.L. Fong, Y.G. Liu, J.C. Martinez, I. Rodero, S.M. Sadjadi, and D. Villegas. Enabling Interoperability among Meta-Schedulers. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 306–315, Lyon, France, May 2008.

[BL98] Amnon Barak and Oren La'adan. The MOSIX multicomputer operating system for high performance cluster computing. *Future Gener. Comput. Syst.*, 13:361–372, March 1998.

[BRC10] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. Intercloud: utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'10, pages 13–31, Berlin, Heidelberg, 2010. Springer-Verlag.

[BW97] F. Berman and R. Wolski. The AppLeS Project: A Status Report. In *Proceedings of the 8th NEC Research Symposium*, May 1997.

[BXM$^+$10] Ilia Baldine, Yufeng Xin, Anirban Mandal, Chris Heermann Renci, Unc-Ch Jeff Chase, Varun Marupadi, Aydan Yumerefendi, and David Irwin. Networked cloud orchestration: A geni perspective. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 573 –578, 2010.

[CCF$^+$94] Karen Castagnera, Doreen Cheng, Rod Fatoohi, Edward Hook, Bill Kramer, Craig Manning, John Musch, Charles Niggley, William Saphir, Douglas Sheppard, Merritt Smith, Ian Stockdale, Shaun Welch, Rita Williams, and David Yip. Clustered workstations and their potential role as high speed compute processors. Technical Report RNS-94-003, NAS Systems Division, NASA Ames Research Center, April 1994.

[CFK^{+}98] Karl Czajkowski, Ian T. Foster, Nicholas T. Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, London, UK, 1998. Springer-Verlag.

[CGGK11] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy H. Katz. The case for evaluating mapreduce performance using workload suites. In *MASCOTS*, pages 390–399, 2011.

[CIG^{+}03] 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 *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 90 – 100, june 2003.

[CNJ^{+}07] Xingchen Chu, Krishna Nadiminti, Chao Jin, Srikumar Venugopal, and Rajkumar Buyya. Aneka: Next-generation enterprise grid platform for e-science and e-business applications. In *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '07, pages 151–159, Washington, DC, USA, 2007. IEEE Computer Society.

[CRCC11] Agustn Caminero, Omer Rana, Blanca Caminero, and Carmen Carrin. Network-aware heuristics for inter-domain meta-scheduling in grids. *Journal of Computer and System Sciences*, 77(2):262 – 281, 2011. ¡ce:title¿Adaptivity in Heterogeneous Environments¡/ce:title¿.

[CTVP10] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. How to enhance cloud architectures to enable cross-federation. In *3rd IEEE International Conference on Cloud Computing (IEEE Cloud 2010)*, pages 337–345, Miami, FL, USA, 2010.

[dAdCB09] Marcos Dias de Assuncao, Alexandre di Costanzo, and Rajkumar Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *HPDC*, pages 141–150, 2009.

[DBC03] Holly Dail, Fran Berman, and Henri Casanova. A decoupled scheduling approach for grid application development environments. *J. Parallel Distrib. Comput.*, 63(5):505–524, May 2003.

[DdAaBV08] Marcos Dias de Assunção, Rajkumar Buyya, and Srikumar Venugopal. Intergrid: a case for internetworking islands of grids. *Concurr. Comput. : Pract. Exper.*, 20(8):997–1024, June 2008.

[DRM] Distributed Resource Management Application API Specifications.

[DSL$^+$08] Ewa Deelman, Gurmeet Singh, Miron Livny, G. Bruce Berriman, and John Good. The cost of doing science on the cloud: the montage example. In *SC*, page 50, 2008.

[EAB$^+$04] T. Eilam, K. Appleby, J. Breh, G. Breiter, H. Daur, S. A. Fakhouri, G. D. H. Hunt, T. Lu, S. D. Miller, L. B. Mummert, J. A. Pershing, and H. Wagner. Using a utility computing framework to develop utility systems. *IBM Syst. J.*, 43(1):97–120, January 2004.

[EC2] Amazon ec2. `http://aws.amazon.com/ec2`.

[EGB03] Lieven Eeckhout, Andy Georges, and Koen De Bosschere. Selecting a reduced but representative workload. In *Middleware Benchmarking: Approaches, Results, Experiences. OOSPLA workshop*, 2003.

[ES01] Dietmar W. Erwin and David F. Snelling. Unicore: A grid computing environment. In *Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, Euro-Par '01, pages 825–834, London, UK, UK, 2001. Springer-Verlag.

[FDF03] R.J. Figueiredo, P.A. Dinda, and J.A.B. Fortes. A case for grid computing on virtual machines. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 550 – 559, may 2003.

[Fei08] D. Feitelson. Parallel Workloads Archive Web Site, 2008.

[FK96] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Perfomance Computing Applications*, 15(3):200–222, 2001.

[For93] Message Passing Interface Forum. Document for a standard message-passing interface. Technical report, Knoxville, TN, USA, 1993.

[FVdW02] Michael Frumkin and Rob F. Van der Wijngaart. Nas grid benchmarks: A tool for grid space exploration. *Cluster Computing*, 5(3):247–255, July 2002.

[FZRL08] I. Foster, Yong Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1 –10, nov. 2008.

[GAE] Google app engine. `http://code.google.com/appengine`.

[Gen01] W. Gentzsch. Sun grid engine: towards creating a compute power grid. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 35 –36, 2001.

[GFKH99] A. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Wide area computing: resource sharing on a large scale. *Computer*, 32(5):29 –37, may 1999.

[GG11] S. Genaud and J. Gossa. Cost-wait trade-offs in client-side resource provisioning with elastic clouds. In *IEEE CLOUD*, pages 1 –8, july 2011.

[GGHL$^+$96] Al Geist, William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing L. Lusk, William Saphir, Anthony Skjellum, and Marc Snir. Mpi-2: Extending the message-passing interface. In *Euro-Par, Vol. I*, pages 128–135, 1996.

[GHJ$^+$09] Albert G. Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *SIGCOMM*, pages 51–62, 2009.

[GHLL$^+$98] William D Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, and Marc Snir. *MPI: the complete reference.* Scientific and Engineering Computation Series. The MIT Press, Cambridge, MA, 1998.

[Gma] Gmail. `http://www.gmail.com`.

[GWT$^+$08] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *SIGCOMM*, pages 75–86, 2008.

[Har75] John A. Hartigan. *Clustering algorithms [by] John A. Hartigan.* Wiley New York,, 1975.

[Hen95] Robert Henderson. Job scheduling under the portable batch system. In Dror Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 279–294. Springer Berlin / Heidelberg, 1995.

[HGR09] David Hadas, Sergey Guenender, and Benny Rochwerger. Virtual Network Services For Federated Cloud Computing. Technical Report H-0269, IBM, 2009.

[HKZ$^+$11] Benjamin Hindman, Andrew Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy H. Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, 2011.

[HML04] E. Huedo, R.S. Montero, and I.M. Llorente. A Framework for Adaptive Execution in Grids. *Softwareractice & Experience*, 34:631–651, June 2004.

[HML07] Eduardo Huedo, Rubn S. Montero, and Ignacio M. Llorente. A modular meta-scheduling architecture for interfacing with pre-ws and ws grid resource management services. *Future Generation Computer Systems*, 23(2):252 – 261, 2007.

[HML10] E. Huedo, R.S. Montero, and I.M. Llorente. Grid architecture from a metascheduling perspective. *Computer*, 43(7):51 –56, july 2010.

[ICG$^+$06] David Irwin, Jeffrey Chase, Laura Grit, Aydan Yumerefendi, David Becker, and Kenneth G. Yocum. Sharing networked resources with brokered leases. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 18–18, Berkeley, CA, USA, 2006. USENIX Association.

[IE11] Alexandru Iosup and Dick H. J. Epema. Grid computing workloads. *IEEE Internet Computing*, 15(2):19–26, 2011.

[II$^+$05] Alexandru Iosup, , Ru Iosup, Dick H. J. Epema, Jason Maassen, and Rob Van Nieuwpoort. Synthetic grid workloads with ibis, koala, and grenchmark. In *In Proceedigs of the CoreGRID Integrated Research in Grid Computing*, 2005.

[IJSE07] Alexandru Iosup, Mathieu Jan, Ozan Sonmez, and Dick Epema. The characteristics and performance of groups of jobs in grids. In Anne-Marie Kermarrec, Luc Boug, and Thierry Priol, editors, *Euro-Par 2007 Parallel Processing*, volume 4641 of *Lecture Notes in Computer Science*, pages 382–393. Springer Berlin / Heidelberg, 2007.

[ILJ$^+$08] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D.H.J. Epema. The Grid Workloads Archive. *Future Generation Computer Systems*, 24:672–686, May 2008.

[IÖF96] Michael A. Iverson, Füsun Özgüner, and Gregory J. Follen. Run-time statistical estimation of task execution times for heterogeneous distributed computing. In *HPDC*, pages 263–, 1996.

[IOY$^+$11] Alexandru Iosup, Simon Ostermann, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick H. J. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE TPDS*, 22(6):931–945, 2011.

[ITF$^+$08a] Alexandru Iosup, Todd Tannenbaum, Matthew Farrellee, Dick Epema, and Miron Livny. Inter-operating grids through delegated matchmaking. *Sci. Program.*, 16(2-3):233–253, April 2008.

[ITF$^+$08b] Alexandru Iosup, Todd Tannenbaum, Matthew Farrellee, Dick H. J. Epema, and Miron Livny. Inter-operating grids through delegated matchmaking. *Scientific Programming*, 16(2-3):233–253, 2008.

[IYE11] A. Iosup, N. Yigitbasi, and D. Epema. On the performance variability of production cloud services. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 104 –113, may 2011.

[JHFS01] D.B. Jackson, B.D. Haymore, J.C. Facelli, and Q.O. Snell. Improving cluster utilization through set based allocation policies. In *Parallel Processing Workshops, 2001. International Conference on*, pages 355 –360, 2001.

[JPC09] Dejun Jiang, Guillaume Pierre, and Chi-Hung Chi. Ec2 performance analysis for resource provisioning of service-oriented applications. In *ICSOC/ServiceWave Workshops*, pages 197–207, 2009.

[JX03] Xuxian Jiang and Dongyan Xu. Soda: a service-on-demand architecture for application service hosting utility platforms. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 174 – 183, june 2003.

[KBM02] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems. *Software Practice and Experience*, 32:135–164, 2002.

[KF99] Nirav H. Kapadia and José A. B. Fortes. PUNCH: An architecture for Web-enabled wide-area network-computing. *Cluster Computing*, 2:153–164, April 1999.

[KFFZ05] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Sci. Program.*, 13(4):265–275, October 2005.

[KP11] Hyunjoo Kim and Manish Parashar. *CometCloud: An Autonomic Cloud Engine*, pages 275–297. John Wiley and Sons, Inc., 2011.

[KTMF09] Katarzyna Keahey, Mauricio Tsugawa, Andrea Matsunaga, and Jose Fortes. Sky computing. *IEEE Internet Computing*, 13:43–51, 2009.

[KV10] Ekasit Kijsipongse and Sornthep Vannarat. Autonomic resource provisioning in rocks clusters using eucalyptus cloud computing. In *MEDES*, pages 61–66, 2010.

[LBN$^+$10] Kien Le, R. Bianchini, T.D. Nguyen, O. Bilgir, and M. Martonosi. Capping the brown energy consumption of internet services at low cost. In *2010 International Green Computing Conference*, pages 3–14, Chicago, IL, USA, 2010.

[LHAP06] Jiuxing Liu, Wei Huang, Bulent Abali, and Dhabaleswar K. Panda. High performance vmm-bypass i/o in virtual machines. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 3–3, Berkeley, CA, USA, 2006. USENIX Association.

[LHL09] Katia Leal, Eduardo Huedo, and Ignacio M. Llorente. A decentralized model for scheduling independent tasks in federated grids. *Future Generation Computer Systems*, 25(8):840 – 852, 2009.

[LJE$^+$10] Wei Lu, J. Jackson, J. Ekanayake, R.S. Barga, and N. Araujo. Performing large science experiments on azure: Pitfalls and solutions. In *CloudCom*, pages 209 –217, 30 2010-dec. 3 2010.

[LKN$^+$09] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *ICSE Workshop on Software Engineering Challenges of Cloud Computing*, May 2009.

[LLM88] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104 –111, jun 1988.

[Loa] Ibm tivoli workload scheduler loadleveler. `http://www-306.ibm.com/software/tivoli/products/scheduler-loadleveler/`.

[LVB$^+$09] Yanbin Liu, David Villegas, Norman Bobroff, Liana Fong, Ivan Rodero, Seetharami Seelam, and S. Masoud Sadjadi. An experimental system for grid meta-broker evaluation. In *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*, LSAP '09, pages 11–18, New York, NY, USA, 2009. ACM.

[MDG$^+$04] J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang. Reseach and Forecast Model: Software Architecture and Performance. In *11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, pages 156–168, Reading, UK, October 2004.

[ME08] Hashim Mohamed and Dick Epema. Koala: a co-allocating grid scheduler. *Concurrency and Computation: Practice and Experience*, 20(16):1851–1876, 2008.

[MF01] Ahuva W. Mu'alem and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12:529–543, 2001.

[MKF10] Paul Marshall, Kate Keahey, and Tim Freeman. Elastic site: Using clouds to elastically extend site resources. *CCGRID*, pages 43–52, 2010.

[MKFG09] M.A. Murphy, B. Kagey, M. Fenn, and S. Goasguen. Dynamic provisioning of virtual organization clusters. In *CCGRID*, pages 364 –371, may 2009.

[MP02] Vijay Mann and Manish Parashar. Engineering an interoperable computational collaboratory on the grid. *Concurrency and Computation: Practice and Experience*, 14(13-15):1569–1593, 2002.

[MvH04] Deborah L. Mcguinness and Frank van Harmelen. OWL web ontology language overview. W3C recommendation, W3C, February 2004.

[MWZS09] Juan C. Martinez, Lixi Wang, Ming Zhao, and S. Masoud Sadjadi. Experimental study of large-scale computing on virtualized resources. In *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, pages 35–42, Barcelona, Spain, 2009.

[NWG+09] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pages 124 –131, may 2009.

[OCC] Occi. `http://ogf.org/documents/GFD.184.pdf`.

[Ope] Opennebula. `http://www.opennebula.com`.

[OTG+05] A. Oleksiak, A. Tullo, P. Graham, T. Kuczynski, J. Nabrzyski, D. Szejnfeld, and T. Sloan. HPC-Europa: Towards Uniform Access to European HPC Infrastructures. In *IEEE/ACM International Workshop on Grid Computing*, pages 308–311, Seattle, WA, USA, November 2005.

[PACR03] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, January 2003.

[PCI] Pci-sig: Peripheral component interconnect special interest group. single root i/o virtualization and sharing 1.1. spec... `http://www.pcisig.com/specifications/iov/single_root/`.

[Rac] Rackspace. `http://www.rackspace.com`.

[Rap04] M. A. Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32 –42, 2004.

[RB08] Rajiv Ranjan and Rajkumar Buyya. Decentralized overlay for federation of enterprise clouds. *CoRR*, abs/0811.2563, 2008.

[RBE+11] Benny Rochwerger, David Breitgand, Amir Epstein, David Hadas, Irit Loy, Kenneth Nagin, Johan Tordsson, Carmelo Ragusa, Massimo Villari, Stuart Clayman, Eliezer Levy, Alessandro Maraschini, Philippe Massonet, Henar Munoz, and Giovanni Toffetti. Reservoir - when one cloud is not enough. *Computer*, 44:44–51, 2011.

[RBL+09] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4:1 –4:11, july 2009.

[RGC+08a] I. Rodero, F. Guim, J. Corbalan, L.L. Fong, Y.G. Liu, and S.M. Sadjadi. Looking for an Evolution of Grid Scheduling: Meta-brokering. *Grid Middleware and Services: Challenges and Solutions*, pages 105–119, August 2008.

[RGC+08b] Ivan Rodero, Francec Guima, Julita Corbalan, Liana Fong, and **S. Masoud Sadjadi**. Evaluation of broker selection strategies. Technical Report UPC-DAC-RR-CAP-2008-41, Computer Architecture Department, Technical University of Catalonia, Barcelona, Spain, Dec. 2008.

[RGC+09] I. Rodero, F. Guim, J. Corbalan, L. Fong, and S. Masoud Sadjadi. Broker selection strategies in interoperable grid systems. In *Parallel Processing, 2009. ICPP '09. International Conference on*, pages 180 –187, sept. 2009.

[RIG+06] Lavanya Ramakrishnan, David Irwin, Laura Grit, Aydan Yumerefendi, Adriana Iamnitchi, and Jeff Chase. Toward a doctrine of containment: grid hosting with adaptive resource control. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[RMNC06] Ala Rezmerita, Tangui Morlier, Vincent Nri, and Franck Cappello. Private virtual cluster: Infrastructure and protocol for instant grids. In *Euro-Par*, volume 4128 of *Lecture Notes in Computer Science*, pages 393–404. Springer, 2006.

[SAH$^+$09] H.E. Schaffer, S.F. Averitt, M.I. Hoit, A. Peeler, E.D. Sills, and M.A. Vouk. Ncsu's virtual computing lab: A cloud computing solution. *Computer*, 42(7):94 –97, july 2009.

[Sal] Salesforce. `http://www.salesforce.com`.

[SAL$^+$04] Jahanzeb Sherwani, Nosheen Ali, Nausheen Lotia, Zahra Hayat, and Rajkumar Buyya. Libra: a computational economy-based job scheduling system for clusters. *Softw., Pract. Exper.*, 34(6), 2004.

[SBS$^+$95] Thomas Sterling, Donald J. Becker, Daniel Savarese, John E. Dorband, Udaya A. Ranawake, and Charles V. Packer. Beowulf: A parallel workstation for scientific computation. In *In Proceedings of the 24th International Conference on Parallel Processing*, pages 11–14. CRC Press, 1995.

[Sch04] Jennifer M. Schopf. Grid resource management. chapter Ten actions when Grid scheduling: the user as a Grid scheduler, pages 15–23. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[SFB$^+$08] S. Masoud Sadjadi, Liana Fong, Rosa M. Badia, Javier Figueroa, Javier Delgado, and *et al.* Transparent grid enablement of weather research and forecasting. In *Proceedings of the 15th ACM Mardi Gras conference*, Baton Rouge, Louisiana, USA, January 2008.

[SGE] Sun grid engine gt4 adaptor. `http://www.lesc.ic.ac.uk/projects/SGE-GT4.html`.

[SKF08] Borja Sotomayor, Kate Keahey, and Ian Foster. Combining batch execution and leasing using virtual machines. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 87–96, New York, NY, USA, 2008.

[SKR03] Gerald Sabin, Rajkumar Kettimuthu, and Arun Rajan. Scheduling of parallel jobs in a heterogeneous multi-site environment. In *in the Proc. of the 9th International Workshop on Job Scheduling Strategies* *for Parallel Processing, Lecture Notes In Computer Science*, pages 87–104, 2003.

[SMLF08] B. Sotomayor, R. Santiago Montero, I. Martin Llorente, and I. Foster. Capacity leasing in cloud systems using the opennebula engine. In *Workshop on Cloud Computing and its Applications 2008 (CCA08)*, Chicago, IL, USA, 2008.

[SMLF09] Borja Sotomayor, Ruben S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13:14–22, 2009.

[Sun90] V. S. Sunderam. PVM: a framework for parallel distributed computing. *Concurrency: Pract. Exper.*, 2:315–339, November 1990.

[TF06] M. Tsugawa and J.A.B. Fortes. A virtual network (vine) architecture for grid computing. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 10 pp., april 2006.

[TMC$^+$08] D. T. Tran, S. Mohan, E. Choi, S. Kim, and P. Kim. A taxonomy and survey on distributed file systems. In *NCM (1)*, pages 144–149, 2008.

[TMMVL12] Johan Tordsson, Rubn S. Montero, Rafael Moreno-Vozmediano, and Ignacio M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2):358 – 367, 2012.

[TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[VASI11] David Villegas, Athanasios Antoniou, Seyed Masoud Sadjadi, and Alexandru Iosup. An analysis of provisioning and allocation policies for IaaS clouds. Technical Report PDS-2011-009, TU Delft, November 2011.

[VASI12] David Villegas, Athanasios Antoniou, Seyed Masoud Sadjadi, and Alexandru Iosup. An Analysis of Provisioning and Allocation Policies for Infrastructure-as-a-Service Clouds. In *CCGRID*, 2012. To appear.

[VBR$^+$12] David Villegas, Norman Bobroff, Ivan Rodero, Javier Delgado, Yanbin Liu, Aditya Devarakonda, Liana Fong, S. Masoud Sadjadi, and Manish Parashar. Cloud federation in a layered service model. *Journal of Computer and System Sciences*, (0):–, 2012.

[VD02] S.S. Vadhiyar and J.J. Dongarra. A metascheduler for the grid. In *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pages 343 – 351, 2002.

[VRF$^+$10] David Villegas, Ivan Rodero, Liana Fong, Norman Bobroff, Yanbin Liu, Manish Parashar, and S. Masoud Sadjadi. *Handbook of Cloud Computing*, chapter The Role Of Grid Computing Technologies in Cloud Computing, pages 183–218. Springer, 2010. (in press; 35 pages; single-spaced).

[VYW$^+$02] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, December 2002.

[WDGK$^+$08] N. Wilkins-Diehr, D. Gannon, G. Klimeck, S. Oster, and S. Pamidighantam. Teragrid science gateways and their impact on science. *Computer*, 41(11):32 –41, nov. 2008.

[WLS$^+$02] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, December 2002.

[YBDS08] L. Youseff, M. Butrico, and D. Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1 –10, nov. 2008.

[YIEO09] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann. C-meter: A framework for performance analysis of computing clouds. In *CCGRID*, pages 472 –477, may 2009.

[Zim81] Hubert Zimmermann. The iso reference model for open systems interconnection. In *Kommunikation in Verteilten Systemen*, pages 39–57, 1981.

VITA

DAVID VILLEGAS

August 4, 1981	Born, Terrassa, Barcelona, Spain
2005	B.A., Computer Engineering Autonomous University of Barcelona Sabadell, Spain

PUBLICATIONS AND PRESENTATIONS

David Villegas, Athanasios Antoniou, S. Masoud Sadjadi and Alexandru Iosup. *An Analysis of Provisioning and Allocation Policies for Infrastructure-as-a-Service Clouds.* Proceedings of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)

David Villegas, Norman Bobroff, Ivan Rodero, Javier Delgado, Yanbin Liu, Aditya Devarakonda, Liana Fong, S. Masoud Sadjadi and Manish Parashar. *Cloud Federation in a Layered Service Model.* In Journal of Computer and System Sciences, January 2012 ISSN 0022-0000, 10.1016/j.jcss.2011.12.017.

David Villegas and S. Masoud Sadjadi. *Mapping Non-Functional Requirements to Cloud Applications.* In Proceedings of the 2011 International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), pages 527-532, Miami, Florida, July 2011.

David Villegas and S. Masoud Sadjadi. *DEVA: Distributed Ensembles of Virtual*

Appliances in the Cloud. In Proceedings of the 17th Euro-Par Conference (Euro-Par 2011), pages 467-478, Bordeaux, France, 2011.

David Villegas, Ivan Rodero, Liana Fong, Norman Bobroff, Yanbin Liu, Manish Parashar, and S. Masoud Sadjadi. Handbook of Cloud Computing, chapter *The Role of Grid Computing Technologies in Cloud Computing.* Springer, 2010 (in press; 36 pages; single-spaced).

Yanbin Liu, David Villegas, Norman Bobroff, Liana Fong, Ivan Rodero, Seetharami Seelam, and S. Masoud Sadjadi. *An experimental system for grid meta-broker evaluation.* In Proceedings of the ACM Large-scale System and Application Performance workshop (LSAP2009) of the International Symposium on High Performance Distributed Computing (HPDC 2009), pages 11-18, Munich, Germany, June 2009.

Norman Bobroff, Liana Fong, Selim Kalayci, Yanbin Liu, Juan Carlos Martinez, Ivan Rodero, S. Masoud Sadjadi, and David Villegas. *Enabling interoperability among meta-schedulers.* In Proceedings of 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid-2008), pages 306-315, Lyon, France, 2008.