1

Cloud Bursting: Managing Peak Loads by Leasing Public Cloud Services

Michael Mattess

The University of Melbourne, Australia

Christian Vecchiola

The University of Melbourne, Australia

Saurabh Kumar Garg

The University of Melbourne, Australia

Rajkumar Buyya

The University of Melbourne, Australia

CONTENTS

In general, small and medium-scale enterprises (SMEs) face problems of unpredictable IT service demand and infrastructure cost. Thus, the enterprises strive towards an IT delivery model which is both dynamic and flexible, and able to be easily aligned with their constantly changing business needs. In this context, Cloud computing has emerged as new approach allowing anyone to quickly provision a large IT infrastructure that can be completely cus-

tomized to the user's needs on a pay-per-use basis. This paradigm opens new perspectives on the way in which enterprises' IT needs are managed. Thus, a growing number of enterprises are out-sourcing a significant percentage of their infrastructure to Clouds.

However from the SMEs perspective, there is still a barrier to Cloud adoption, being the need of integrating current internal infrastructure with Clouds. They need strategies for growing IT infrastructure from inside and selectively migrate IT services to external Clouds in a way that enterprises benefit from both Cloud infrastructure's flexibility and agility as well as lower costs.

In this chapter, we present how to profitably use Cloud computing technology by using Aneka, which is a middleware platform for deploying and managing the executions of applications on different Clouds. This chapter also presents public resource provisioning policies for dynamically extending the enterprise IT infrastructure to evaluate the benefit of using public Cloud services. This technique of extending capabilities of enterprise resources by leasing public Cloud capabilities is also known as Cloud bursting. The policies rely on a dynamic pool of external resources hired from commercial IaaS providers in order to meet peak demand requirements. To save on hiring costs, hired resources are released when they are no longer required. The described policies vary in the threshold used to decide when to hire and when to release; the threshold metrics investigated are queue length, queue time as well as a combination of these. Results demonstrating that simple thresholds can provide an improvement to the peak queue times, hence keeping the system performance acceptable during peaks in demand.

1.1 Introduction

Recently there has been a growing interest in moving infrastructure, software applications and hosting of services from in-house server rooms to external providers. This way of making IT resources available, known as Cloud computing, opens new opportunities to small, medium, and large sized companies. It is not necessary any more to bear considerable costs for maintaining the IT infrastructures or to plan for peak demand, but infrastructure and applications can scale elastically according to the business needs at a reasonable price. The possibility of instantly reacting to the demand of customers without long term infrastructure planning is one of the most appealing features of Cloud computing and it has been a key factor in making this trend popular among technology and business practitioners. As a result of this growing interest, the major players in the IT playground such as Google, Amazon, Microsoft, Oracle, and Yahoo, have started offering Cloud computing based solutions that cover the entire IT computing stack, from hardware to applications and services. These offerings have quickly become popular and led to

the establishment of the concept of the "Public Cloud", which represents a publicly accessible distributed system hosting the execution of applications and providing services billed on a pay-per-use basis.

Because Cloud computing is built on a massively scalable shared infrastructure, Cloud suppliers can in theory quickly provide the capacity required for very large applications without long lead times. Purchasers of Infrastructure as a Service (IaaS) capacity can run applications on a variety of virtual machines (VMs), with flexibility in how the VMs are configured. Some Cloud computing service providers have developed their own ecosystem of services and service providers that can make the development and deployment of services easier and faster. Adding SaaS capacity can be as easy as getting an account on a supplier's website. Cloud computing is also appealing when we need to quickly add computing capacity to handle a temporary surge in requirements. Rather than building additional infrastructure, Cloud computing could in principle be used to provide on-demand capacity when needed. Thus, the relatively low upfront cost of IaaS and PaaS services, including VMs, storage, and data transmission, can be attractive. Especially for addressing tactical, transient requirements such as unanticipated workload spikes. An additional advantage is that businesses pay only for the resources reserved; there is no need for capital expenditure on servers or other hardware.

However, despite these benefits, it has become evident that a solution built on outsourcing the entire IT infrastructure to third parties would not be applicable in many cases. On the one hand, enterprise applications are often faced with stringent requirements in terms of performance, delay, and service uptime. On the other hand, little is known about the performance of applications in the Cloud, the response time variation induced by network latency, and the scale of applications suited for deployment; especially when there are mission critical operations to be performed and security concerns to consider. Moreover, with the public Cloud distributed anywhere on the planet, legal issues arise and they simply make it difficult to rely on a virtual public infrastructure for some IT operation. In addition, enterprises already have their own IT infrastructures, which they have been using so far.

In spite of this, the distinctive feature of Cloud computing still remains appealing to host part of applications on in-house infrastructure and others can be outsourced. In other words, external Clouds will play a significant role in delivering conventional enterprise compute needs, but the internal Cloud is expected to remain a critical part of the IT infrastructure for the foreseeable future. Key differentiating applications may never move completely out of the enterprise because of their mission-critical or business-sensitive nature. Such infrastructure is also called a hybrid Cloud which is formed by combining both private and public Clouds whenever private/local resources are overloaded. The notion of hybrid Clouds that extend the capabilities of enterprise infrastructure by leasing extra capabilities from public Clouds is also known as Cloud bursting.

For this vision to be achieved, however, both software middleware and

scheduling policies supporting provisioning of resources from both local infrastructures and public Clouds are required. So that applications can automatically and transparently expand into public virtual infrastructures. The software middleware should offer enterprises flexibility in decision making that can enable them to find the right balance between privacy considerations, performance and cost savings. Thus, in this chapter we first describe the architecture of a Cloud middleware called Aneka, which is a software platform for building and managing a wide range of distributed systems. It allows applications to use resources provisioned from different sources, such as private and public IaaS Clouds. Such a hybrid system built with resources from a variety of sources are managed transparently by Aneka. Therefore, this platform is able to address the requirements to enable execution of compute intensive applications in hybrid Clouds.

In summary, in this chapter, we will present the architecture of the Aneka middleware for building hybrid Clouds by dynamically growing the number of resources during periods of peak demand. We will then propose and evaluate three scheduling approaches to manage the external pool of resources and achieve cost benefits for enterprises.

1.2 Aneka

Aneka [12] is a software platform and a framework for developing distributed applications in the Cloud harnessing the computing resources of a heterogeneous network. It can utilise a mix of resources such as workstations, clusters, grids and servers in an on demand manner. Aneka implements a Platform as a Service model, providing developers with APIs for transparently exploiting multiple physical and virtual resources in parallel. In Aneka, application logic is expressed with a variety of programming abstractions and a runtime environment on top of which applications are deployed and executed. System administrators leverage a collection of tools to monitor and control the Aneka Cloud, which can consist of both company internal (virtual) machines as well as resources form external IaaS providers.

The core feature of the framework is its service oriented architecture that allows customization of each Aneka Cloud according to the requirements of users and applications. Services are also the extension point of the infrastructure: by means of services it is possible to integrate new functionalities and to replace existing ones with different implementations. In this section we briefly describe the architecture and categorize the fundamental services that build the infrastructure.

Figure 1.1 provides a layered view of the framework. Aneka provides a runtime environment for executing applications by leveraging the underlying infrastructure of the Cloud. Developers express distributed applications

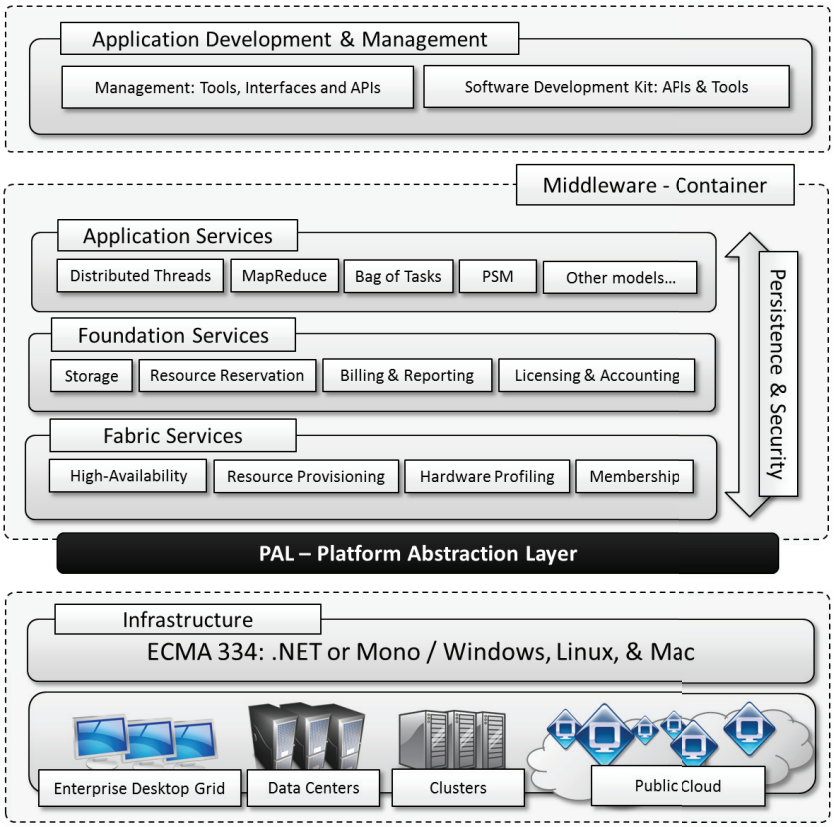


FIGURE 1.1
The Aneka Framework

by using the APIs contained in the Software Development Kit (SDK). Such applications are executed on the Aneka Cloud, consisting of a collection of worker nodes hosting the Aneka Container. The Container is the core building block of the middleware and can host a number of services. These include the runtime environments for the different programming models as well as middleware services. Administrators can configure which services are present in the Aneka Cloud. Using such services for the core functionality provides an extendible and customisable environment. There are three classes of services that characterize the Container:

Execution Services: these services are responsible for scheduling and executing applications. Each programming model supported by Aneka defines specialized implementations of these services for managing the execution of a work unit defined in the model.

Foundation Services: these services are the core management services of the Aneka Container. They are in charge of metering applications, allocating resources for execution, managing the collection of available nodes, and keeping the services registry updated.

Fabric Services: these services constitute the lowest level of the services stack of Aneka and provide access to the resources managed by the Cloud. An important service in this layer is the Resource Provisioning Service, which enables horizontal scaling (e.g, increase and decrease in the number of VMs) in the Cloud. Resource provisioning makes Aneka elastic and allows it to grow and shrink dynamically to meet the QoS requirements of applications

The Container relies on a Platform Abstraction Layer that interfaces with the underlying host, whether this is a physical or a virtualized resources. This makes the Container portable over different platforms that feature an implementation of the ECMA 335 (Common Language Infrastructure) specification. Two well known environments that implement such a standard are the Microsoft .NET framework and the Mono open source .NET framework.

1.3 Hybrid Cloud Deployment Using Aneka

Hybrid deployments constitute one of the most common deployment scenarios of Aneka [13]. In many cases, there is an existing computing infrastructure that can be leveraged to address the computing needs of applications. This infrastructure will constitute the static deployment of Aneka that can be elastically scaled on demand when additional resources are required. An overview of such a deployment is presented in Figure 1.2.

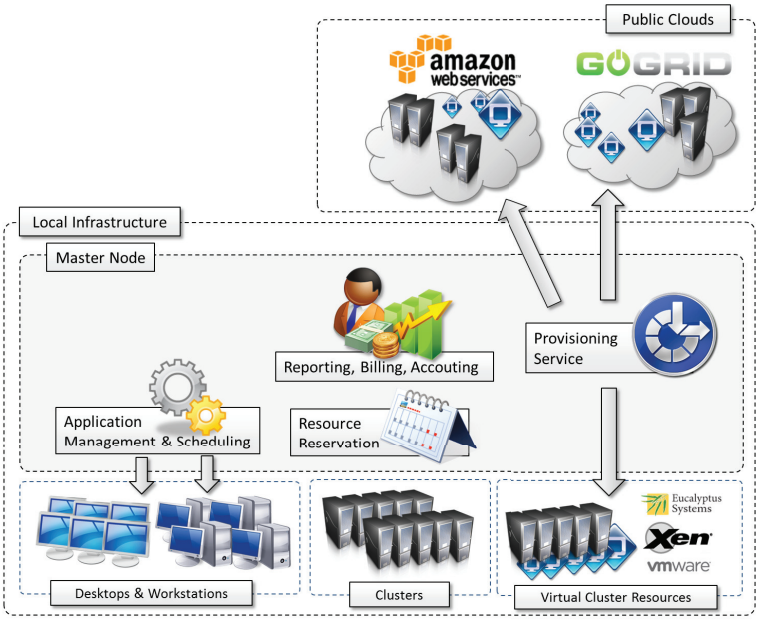


FIGURE 1.2
Aneka Hybrid Cloud

This scenario constitutes the most complete deployment for Aneka which is able to leverage all the capabilities of the framework:

- Dynamic Resource Provisioning.
- Resource Reservation.
- Workload Partitioning.
- Accounting, Monitoring, and Reporting.

In a hybrid scenario heterogeneous resources can be used for different purposes. For example, a computational cluster can be extended with desktop machines, which are reserved for low priority jobs outside the common working hours. The majority of the applications will be executed on these local resources, which are constantly connected to the Aneka Cloud. Any additional demand for computing capability can be leased from external IaaS providers. The decision to acquire such external resources is made by provisioning policies plugged into the scheduling component, as this component is aware of the current system state. The provisioning service then communicates with the IaaS provider to initiate additional resources, which will join the pool of worker nodes.

1.4 Motivation: Case Study Example

Enterprises run a number of HPC applications to support their day-to-day operation. This is evident from the recent Top500 supercomputer applications [10]. where many supercomputers are now used for industrial HPC applications, such as 9.2% of them are used for Finance and 6.2% for Logistic services. Thus, it is desirable for IT industries to have access to a flexible HPC infrastructure which is available on demand with minimum investment. In this section, we explain the requirements of one of such enterprise called GoFront Group, and how they satisfied their needs by using the Aneka Cloud Platform, which allowed them to integrate private and public Cloud resources.

The GoFront Group in China is a leader in research and manufacture of electric rail equipment. Its products include high speed electric locomotives, metro cars, urban transportation vehicles, and motor train sets. The IT department of the group is responsible for providing support for the design and prototyping of these products. The raw designs of the prototypes are required to be rendered to high quality 3D images using the Autodesk rendering software called Maya. By examining the 3D images, engineers are able to identify any potential problems from the original design and make the appropriate changes. The creation of a design suitable for mass production can take many months or even years. The rendering of the three dimensional models is one of

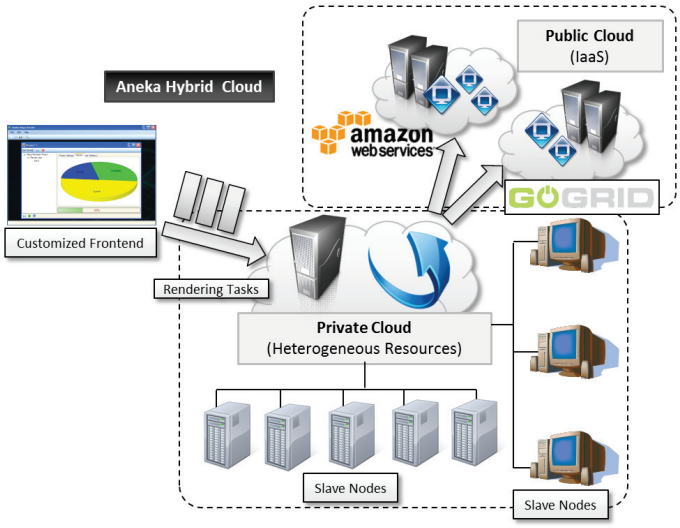


FIGURE 1.3
Aneka Hybrid Cloud Managing GoFront Application

the phases that absorbs a significant amount of time since the 3D model of the train has to be rendered from different point of views and for many frames. A single frame with one camera angle defined can take up to 2 minutes to render. To render one completes set of images from one design takes over 3 days. Moreover, this process has to be repeated every time a change is applied to the model. It is then fundamental for GoFront to reduce the rendering times, in order to be competitive and speed up the design process.

To solve the company's problem, an Aneka Cloud can be set up by using the company's desktop computers and servers. The Aneka Cloud also allows dynamic leasing of resources from a public Cloud in the peak hours or when desktop computers are in use. Figure 1.3 provides an overall view of the installed system. The setup is constituted by a classic master slave configuration in which the master node concentrates the scheduling and storage facilities and thirty slave nodes are configured with execution services. The task programming model has been used to design the specific solution implemented at GoFront. A specific software tool that distributes the rendering of frames in the Aneka Cloud and composes the final rendering has been implemented to help the engineers at GoFront. By using the software, they can select the parameters required for rendering and leverage the computation on the private Cloud.

The rendering phase can be speed up by leveraging the Aneka hybrid Cloud, compared to only using internal resources. Aneka provides a fully integrated infrastructure of the internal and external resources, creating, from

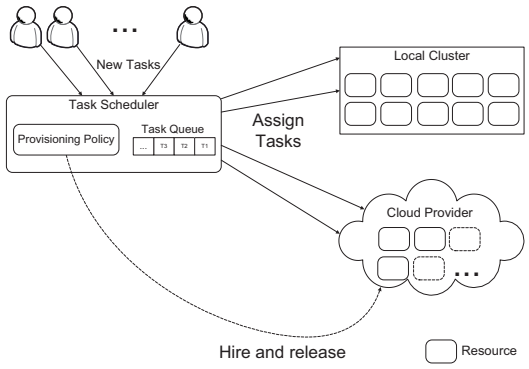


FIGURE 1.4
The task scheduling and resource provisioning scenario.

the users perspective, a single entity. It is however important to manage the usage of external resources as a direct hourly cost is inured. Hence policies are needed that can be configured to respond to the demand on the system by acquiring resources only when it is warranted by the situation. Therefore, we have designed some scheduling policies which consider the queue of waiting tasks when deciding to lease resources. These policies are further described in detail with the analysis of their cost saving benefits in subsequent sections.

1.5 Resource Provisioning Policies

In Aneka, as discussed in the previous section, there are two main components which handle the scheduling of task on the hybrid Cloud. They are the scheduling and provisioning components. The Provisioning component is responsible for the interactions between Aneka and the external resource providers, be they commercial IaaS Cloud or private clusters with a Cloud software stack like Eucalyptus. As each IaaS provider and software stack have their own API, a pluggable driver model is needed. The API specifics are implemented in what are termed resource pools. A number of these pools have been implemented to support commercial providers such as Amazon EC2 and GoGrid. Private Cloud stacks such as Eucalyptus and VMware products are also supported. The scheduling component is responsible for assigning tasks to resources, it keeps track of the available resources and maintains a queue of the tasks waiting to execute. Within this component there is also a pluggable

scheduling and provisioning interface, so that Aneka can be configured to use different scheduling policies. The decision to request more resources or release resources (that are no longer needed) is also made by the selected scheduling policy. A number of such provisioning policies are implemented and can be selected for use.

These provisioning policies are the core feature of Aneka to enable the dynamic acquisition and release of external Cloud resources. As such resources have a cost attached, it is important to maximise their utilisation and at the same time, request them only when they will be of benefit for the system. In the most common scenario, Cloud resources are billed according to time blocks that have a fixed size. In case of peak demands new resources can be provisioned to address the current need, but then used only for a fraction of the block for which they will be billed. Terminating these resources just after their usage could potentially led to a waste of money, whereas they could be possibly reused in the short term and before the time block expires. In order to address this issue, we introduced a resource pool that keeps resources active and ready to be used until the time block expires.

Figure 1.4 describes the interaction between the scheduling algorithm, the provisioning policy, and the resource pool. Tasks in the queue are processed in order of arrival. A set of different events such as the arrival of a new task, its completion, or a timer can trigger the provisioning policy that, according to its specific implementation, will decide whether to request additional resources from the pool or release them. The pool will serve these requests, firstly with already active resources, and if needed by provisioning new ones from the external provider.

The policies described below rely on the common principle of grow and shrink thresholds to determine when additional resources will be of benefit. That is, once a metric passes the defined growth threshold a request for additional resources is triggered. Conversely the shrink threshold is used by the policy to determine if a resource is no longer required.

1.5.1 Queue Length

The *Queue Length* based policy uses the number of tasks that are waiting in the queue as the metric for the grow and shrink thresholds. When a task arrives the number of tasks in the queue is compared to the growth threshold, if the queue length exceeds the threshold an additional resource is requested. When a resource in the external Cloud becomes free the queue length is compared to the shrink threshold. If the number of waiting tasks is less than the shrink threshold the resource is released back to the pool, otherwise it will be used to execute the next task.

Algorithm 1 Queue Length: Task Arrival

```
if Queue.Length ≥ Growth_Threshold then
  return RequestResource
else
  return NoAction
end if
```

Algorithm 2 Queue Length: Task Finished

```
if Queue.Length ≤ Shrink_Threshold then
  return ReleseResource
else
  return KeepResource
end if
```

1.5.2 Queue Time

The *Queue Time* based policy uses the time individual tasks have spent in the queue to determine when additional resources are requested. This policy periodically checks for how long the task at the head of the queue has been waiting. If this time exceeds the growth threshold, additional resources are requested. The number of requested resources is the same as number of tasks that have exceeded the growth threshold. When a resource in the Cloud becomes free, the amount of time the task at the head of the queue has been waiting is compared to the shrink threshold, if it is less than the threshold the resource will be released, otherwise it will be used to execute the next task.

Algorithm 3 Queue Time: Periodic Check

```
requestSize ← 0
index ← 0
while Queue[index].QueueTime ≥ Growth_Threshold do
  requestSize ← requestSize + 1
  index ← index + 1
end while
requestSize ← requestSize − OutstandingResourceCount
if requestSize > 0 then
  RequestResources(requestSize)
end if
```

1.5.3 Total Queue Time

The *Total Queue Time* policy sums the queue time of each of the tasks in the queue, starting from the tail of the queue. When the sum exceeds the growth threshold, before reaching the head of the queue, a resource is requested for

Algorithm 4 Queue Time: Task Finished

```
if Queue[head].QueueTime ≤ Shrink_Threshold then
  return ReleseResource
else
  return KeepResource
end if
```

each of the remaining tasks. The releasing of a resource occurs when a task on a external Cloud resource has finished. At that time the total queue time is established and compared to the shrink threshold. If it is less than the threshold the resource is released, otherwise it will be used for the next task.

Algorithm 5 Total Queue Time: Periodic Check

```
requestSize ← 0
totalQueueTime ← 0
for i = 0 to Queue.Length do
  totalQueueTime ← totalQueueTime + Queue[i].QueueTime
  if totalQueueTime ≥ Growth_Threshold then
    requestSize ← requestSize + 1
  end if
end for
requestSize ← requestSize − OutstandingResourceCount
if requestSize > 0 then
  RequestResources(requestSize)
end if
```

Algorithm 6 Total Queue Time: Task Finished

```
totalQueueTime ← 0
for i = 0 to Queue.Length do
  totalQueueTime ← totalQueueTime + Queue[i].QueueTime
end for
if totalQueueTime < ShrinkThreshold then
  return ReleseResource
else
  return KeepResource
end if
```

1.5.4 Clairvoyant Variant

In our scenario we consider the execution time of a task an unknown until the task finishes. Hence these policies can not know if a particular task will complete before a time block finishes or if charges for additional time blocks

will be incurred. This makes it difficult to utilise the remaining time of external resource, which are being released by the policy. However to explore the impact of better utilising this otherwise wasted time we created clairvoyant variants of these policies. These variants are able to predict the run time of each task in the queue. As the policy decides to release a resource the clairvoyant variants perform a pass over all the tasks in the queue, searching for tasks which will complete within the remaining time. The best fit task is then assigned to the resource, which remains in use. If no task is found, which will complete in time, the resource is released.

Algorithm 7 Clairvoyant Variants: Task Finished

```
remainingTime ← ResourceToBeReleased.RemainingTimeInBlock
bestFitRunTime ← 0
bestFitTask ← NULL
for i = 0 to Queue.Length do
  task ← Queue[i]
  runTime ← task.RunTime
  if runTime ≤ remainingTimeANDrunTime > bestFitRunTime
  then
    bestFitTask ← task
    bestFitRunTime ← runTime
  end if
end for
if bestFitTask ≠ NULL then
  AssignTask(bestFitTask, ResourceToBeReleased)
  return KeepResource
else
  return ReleseResource
end if
```

1.6 Performance Analysis

The policies described in the previous section have been implemented in Aneka and have been used to observe the behaviour of the system for small and real workloads. In order to extensively test and analyse these policies we built a discrete event simulator that mimics the Aneka scheduling interfaces, so that that polecies can be easaly ported between the simulator and Aneka. We also identified a performance metric that will drive the discussion of the experimental results presented in this section.

1.6.1 Simulator

To analyse the characteristics of these provisioning policies it is necessary to construct a simulator. Using a simulator in this case allows policies to be compared with exactly the same workload. It also means that any interruptions to production Aneka environments are avoided. But most importantly the simulations can be run much faster than real-time, so that a whole month or year can be simulated in minutes.

The simulator used here is especially constructed to reflect the internal interfaces of Aneka, so that implementation of scheduling and provisioning policies could be easily ported from the simulator the Scheduling module in Aneka. In the simulator design, the fundamental goal was to gain an insight in to these policies while keeping the design as simple as possible. Hence the simulator does not attempt to capture all the complexities of a distributed environment, instead it focuses on exposing the behaviour of the policies.

1.6.2 Performance Metric

To examine and compare the performance characteristics of the policies we use a metric which we termed the *Top Queue Time Ratio*. This metric comprises the average of the 5000 longest queue times, which is then divided by the average execution time of all the tasks.

We consider the average of the largest queue times rather than an overall average as our focus is on managing the spikes in demand rather then improving the average case. We decided to consider as a reference the top 5000 largest queue times. The reason for this choice is that this number represents about 15% of the workload used for the analysis. In our opinion the acceptability of a waiting time is to some extent relative to the execution time of the task. Hence, we use the ratio between the largest queue times and the average task duration. Using a metric which is relative to the average duration of the tasks also allows us to scale time in the workload without affecting the metric, thus making direct comparisons possible.

1.6.3 Workload

To drive the simulation we use a trace from the Auver Grid [1]. It was chosen as it only consists of independent tasks, which reflects the scenario described in the motivation section (Section 1.4). This trace covers the whole of 2006 and has been analysed in [5]. The Auver Grid is a production research grid located in France including 475 Pentium 4 era CPUs distributed over six sites.

We divided the trace into calendar months to compare periods with high and low demand on the system. As it is clear from Table 1.1, October experienced the greatest load on the system. This is also reflected in the *Top Queue Times* of the months shown in Figure 1.5. Hence, we used the trace of October to carry out the experiments.

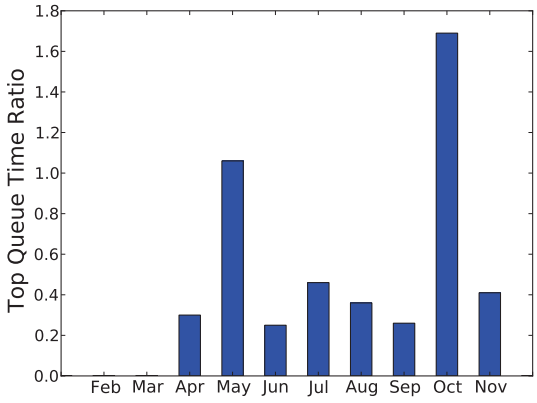


FIGURE 1.5
Top Queue Time Ratio of each month.

	Av.RunTime(min)	Task Count	Total RunTime
Jan	318	10,739	3,415,002
Feb	255	15,870	4,046,850
Mar	242	39,588	9,580,296
Apr	484	27,188	13,158,992
May	493	32,194	15,871,642
Jun	361	35,127	12,680,847
Jul	311	46,535	14,472,385
Aug	605	28,934	17,505,070
Sep	552	30,002	16,561,104
Oct	592	33,839	20,032,688
Nov	625	14,734	9,208,750
Dec	382	24,564	9,383,448

TABLE 1.1
Characteristics of each month

Scaling Factor	1	10	20	40
Av. Task Duration (minutes)	592	59	29	15

TABLE 1.2
The average task duration, during October, for scaling factors used.

The average duration of a task in this workload during the month of October is almost 10 hours. This is a significantly longer time period than the one hour blocks, which form the de-facto atomic unit for buying time in the Cloud. We felt that the relationship between the average task duration and the size of the blocks in which time is bought is worth exploring. Hence, we used a scaling factor to divide the task duration and arrival interval times, essentially compressing time. This reduced the average task duration (see table 1.2) while maintaining the same load on the system. The Top Queue Time Ratio is also left unaffected by the scaling, thus allowing it to be used to compare the results with different scaling factors. The cost, on the other hand, is affected by the scaling as the absolute total amount of task time is scaled down. As less time is required to process tasks less time ends up being bought from the Cloud provider. To allow cost comparisons of different scaling factors we multiply the cost by the scaling factor such that the cost is representative of the entire month rather than the scaled down time.

1.6.4 Experimental Setup

Although our simulations do not intend to replicate the Auver Grid (see Workload Section 1.6.3) we referred to it when configuring the simulated environment. In particular, the number of permanent local resources in the simulation, is set to 475, reflecting the number of CPUs in the Auver Grid. Also, when considering the relative performance between the local and Cloud resources the Pentium 4 era CPUs of the Auver Grid were compared to Amazon's definition of their EC2 Compute Unit [11] and it was concluded that they are approximately comparable.

Hence, a Small EC2 Instance was used as the reference point, which has one EC2 compute unit. Thus for all the simulations the relative performance was configured such that a particular task would take that same amount of time to run locally or in the Cloud. To calculate a cost for the use of Cloud resources we again use a Small EC2 Instance as a reference point with an hourly cost of USD $0.10 [1]. Although the true cost of using Cloud resources includes other costs such as network activity both at the providers end and the local end as well as data storage in the Cloud. We have not attempted to establish these costs as they are dependant on the individual case. Hence, the

[1] Amazon has since reduced its prices.

cost for using Cloud resources in this work is solely based on the amount of billable time.

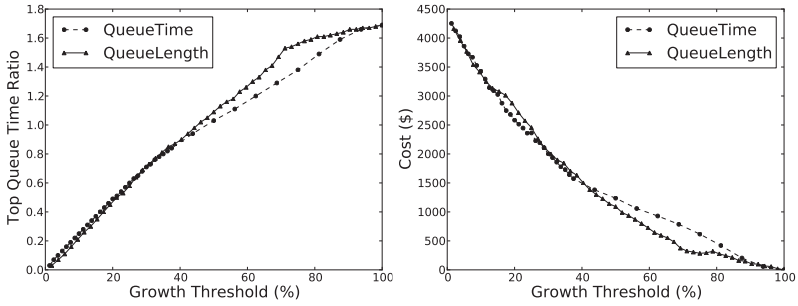
When requesting resources from a Cloud provider, there is generally a *time block* which represents an atomic billable unit of usage. In the case of Amazon's EC2 this time block is one hour. Meaning that if a resource is used for 1.5 hours the cost will be that of two time blocks. This behaviour is reflected in the simulations where the size of the time block is also set to one hour. As requested resources take time to come on line we have set a three minute delay between a resource being requested and it becoming available to execute tasks. Our own usage of EC2 has shown this to be a reasonable amount of time for a Linux instance to become available. Although these times can vary considerably on EC2 we do not anticipate getting a better insight into the policies by introducing random delays to the simulation.

1.6.5 Experimental Results

In this section we explore the effects of the parameters on the performance of the policies. First we examine the effects of the settings associated with the policies. Later we look at the impact the task size has on the efficiency in terms of the unused time of resources in the Cloud.

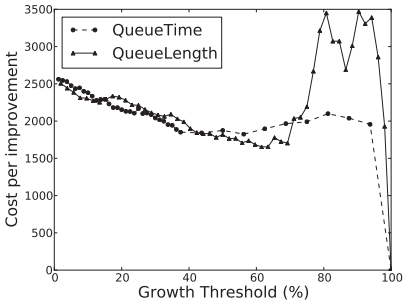
First of all we take a look at the Growth Threshold's impact. A low Growth Threshold means that the policy will request additional resources sooner. As the Growth Threshold increases the burden on the system needed to trigger a request for additional resources also increases. Hence, as the Growth Threshold increases the time some tasks spend waiting in the queue also increases. The graph in Figure 1.6(a) demonstrated this trend showing the Top Queue Time as the Growth Threshold increases. In this figure the Growth Threshold is expressed as a percentage of the maximum Growth Threshold, which is a threshold so high that the workload we are using does not trigger a request for additional resources. Such a threshold has been determined by repeatedly increasing the value set until no more additional resources are requested.

As one would expect the Growth Threshold to cost relationship (see Figure 1.6(b)) is inversely proportional to that of the Top Queue Time Ratio (see Figure 1.6(a)). As a high Top Queue Time Ratio means that less external resources were requested resulting in a lower cost. The graph in Figure 1.6(c) shows the cost per unit reduction of the Top Queue Time Ratio. The most striking feature of this graph are the two spikes of the Queue Length based policy. When the growth threshold reaches the size where these spikes occur there is only one instance of high demand in the workload, which is large enough to causes additional resources to be requested. The Queue Length based policy in this case requests more resources than would be ideal, hence incurring costs, which are not reflected in a reduction of the Top Queue Time Ratio. The Queue Time based policy on the other hand triggers fewer additional resources to be requested, which means that the cost is in proportion to the improvement in the Top Queue Time Ratio. As at that point (around 80%)



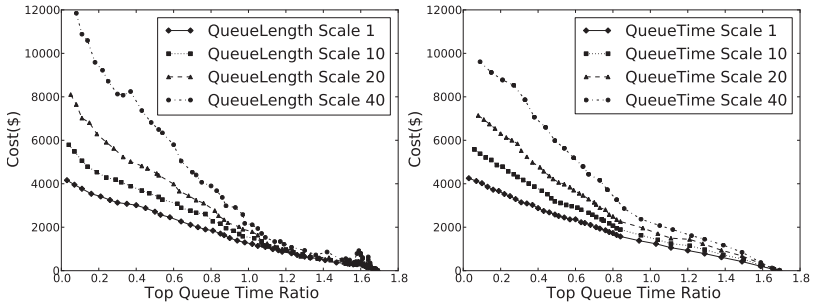
(a) Top Queue Time Ratio

(b) Cost



(c) Top Queue Time Ratio to Cost relationship

FIGURE 1.6
The performance of the Queue Time and Queu Length based policies as the growth threshold increases.



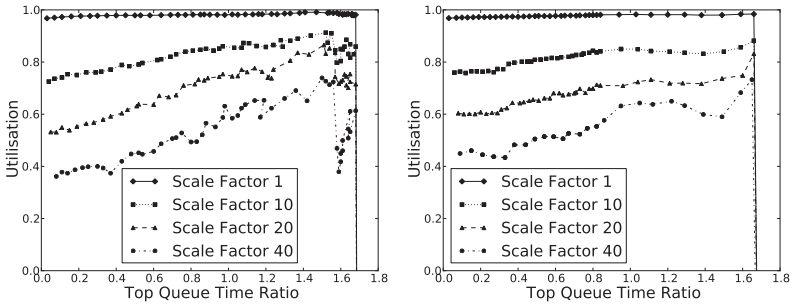
(a) Queue Length Based (b) Queue Time Based

FIGURE 1.7
Demonstrating the relationship between the Top Queue Time and the Cost. The Growth Threshold is varied between the points on a line. The scaling factor is changed between lines.

the cost and improvement in the Top Queue Time Ratio are quite small, the graph comparing these two values becomes more sensitive to small changes, resulting the large size of these spikes. Apart from the spikes Figure 1.6(c) shows that the inverse relationship between the Top Queue Time Ratio and the Cost has a slight downward trend until around the middle of the growth threshold range. Meaning that the best value is with a growth threshold around the middle of the range of values.

Figure 1.6 also demonstrates that the Queue Length and Queue Time based policies behave very similarly when the task size is large compared to the size of the time blocks purchased from the Cloud provider. Comparing the lines of scale factor one in Graph 1.7(a) and 1.7(b) further confirm this by indicating that the cost of achieving a particular Top Queue Time Ratio is virtually identical.

However, as the scale factor is increased the cost of reaching a particular Top Queue Time Ratio also increases in Figure 1.7. In this Figure the cost has been multiplied by the scaling factor, as scaling reduces the total amount of work that needs to be done, which we have to normalise against to be able to compare the results using different scaling factors (see section 1.6.3). Hence a smaller task size increases the amount of time that needs to be purchased from the Cloud to reach a particular Top Queue Time Ratio. Comparing the graphs in Figure 1.7 also shows that the Queue Time based policy performs slightly better at lower Top Queue Time Ratios and that the Queue Length bases policy has a slight edge at higher Top Queue Time Ratios. But these slight differences only become apparent as the tasks become smaller.

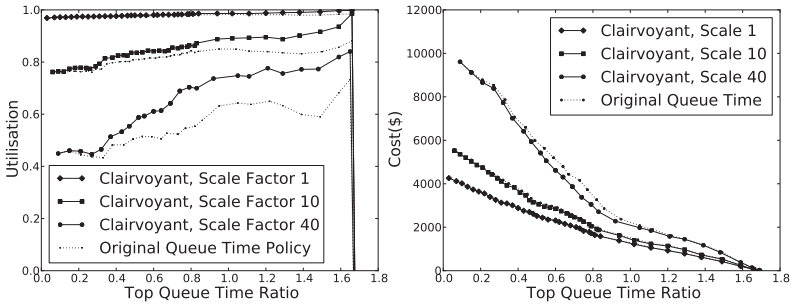


(a) Utilisation - Queue Length Based (b) Utilisation - Queue Time Based

FIGURE 1.8
Shows the utilisation of the time bought on the Cloud. The utilisation is calculated using only time available for executing tasks and not start up times for resources in the Cloud. The Growth Threshold is varied between the points on a line. The scaling factor is changed between lines.

To examine the increased cost with the smaller task size we firstly look at the utilisation of the time that is bought from the Cloud provider in Figure 1.8. We see that with smaller tasks a significant proportion of the available resource time in the Cloud goes unused. Comparing Figures 1.7 and 1.8 it can be noticed that some features correlate between the cost and utilisation. In particular for the Queue Length based policy the lines move closer for larger Top Queue Time Ratios. The Queue Time based policy on the other hand maintains a more consistent separation in both the cost and utilisation graphs. Hence, we attribute the increased cost with smaller tasks to the decreased efficiency with which resources in the Cloud are used.

In Figure 1.9(a) we compare the utilisation of the Queue Time based policy to its clairvoyant variant. This graph is essentially Figure 1.8(b) with the clairvoyant results superimposed. We can see that at very low Top Queue Time Ratios the clairvoyant variant performs only marginally better. At higher queue time ratios the clairvoyant variant is able to significantly improve the utilisation of the time bought in the Cloud. However in Figure 1.9(b), which is derived form Figure 1.7(b) we see that the improvement in utilisation has not lead to a comparable reduction of cost for achieving a particular Top Queue Time Ratio. This is because the clairvoyant policy uses backfilling to utilise otherwise wasted time of resources being released. Hence, it does not directly change the growing of resources in the Cloud. Its effect on cost is only indirect by removing some tasks from the queue, which will effect when the thresholds are crossed. Also our metric focuses on the longest queue times, where as the



(a) Utilisation comparison to original data from Figure 1.8(b)

(b) Cost comparison to original data from Figure 1.7(b)

FIGURE 1.9
Contrasts the Queue Time based policy with its clairvoyant variant. Scale factor 20 is omitted in the interest of clarity.

clairvoyant variant allows more tasks to be processed when resources are being released. Hence the benefit of the clairvoyant variant comes at a time where the queue has already reached an acceptable size and will only have a limited effect on the largest queue times.

Hence, to impact cost we need to be looking at when additional resources are requested. The Total Queue Time policy is in essence a combination of the simpler Queue Length and Queue Time based policies. In Figure 1.10 the cost is compared to that of the Queue Time based policy discussed previously. We see that for lower Top Queue Time Ratios and bigger tasks a clear improvement is possible. But once the average task duration becomes less than the time blocks being purchased in the Cloud the advantage disappears. For small average task durations (scaling factor 40) this policy can perform significantly worse when aiming at higher Top Queue Time Ratios.

Up to now our investigation has focused on the month of October in the 2006 Auver Grid trace. We will now take a look at how both the Queue Time and Queue Length based policies behave in the other months of the trace. For this comparison we configured both policies with a growth threshold value that for October resulted in a Top Queue Time Ratio close to 0.7 and did not apply any scaling of the trace. Figure 1.11 presents the Top Queue Time Ratio and corresponding cost for each month. Clearly during October and May the system is under the greatest load, which is considerably higher than any other month. Hence as expected these policies have their greatest impact during these month. There is however an interesting observations to make. During November both policies trigger requests for additional resources,

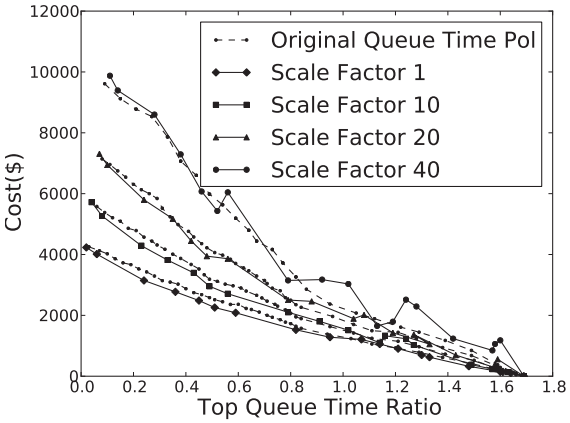


FIGURE 1.10
Contrasts the Queue Time Total policy with the Queue Time policy.

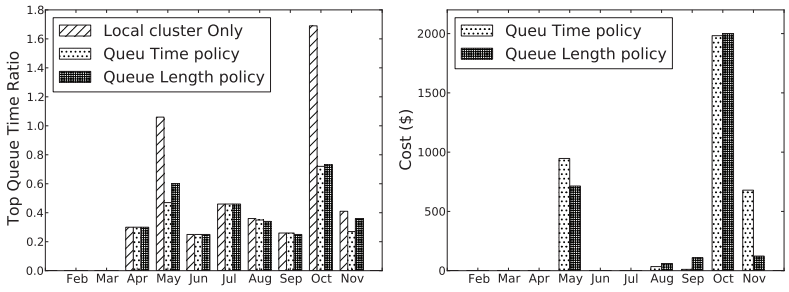
however during July, which has a slightly higher Top Queue Time Ratio, the policies do not trigger any requests.

To understand this behaviour we need to look at the average task duration and task count for each month, which are presented in Table 1.1. First we note that the average task duration of November is almost exactly double that of July. The Top Queue Time Ratio is relative to the average task size. Hence as both July and November have similar Top Queue Time Ratios, the actual top 5000 queue times in November are almost double the length of those in July. The Queue Time based policy hoverer, uses a fixed growth threshold, which leads it to trigger requests for additional resources. The Queue Length based policy is affected to a lesser extent.

1.7 Related Work

This work considers the case where an enterprise wants to provision resources from an external Cloud provider to meet its peak compute demand in the form of Bag-of-Tasks applications. To keep the queue waiting time of HPC tasks at an acceptable level, the enterprise hires resources from public Cloud (IaaS) providers.

A key provider of on-demand public Cloud infrastructure is Amazon Inc. with its Elastic Compute Cloud (EC2) [11]. EC2 allows users to deploy VMs on Amazon's infrastructure, which is composed of several data centres located around the world. To use Amazon's infrastructure, users deploy instances



(a) Top Queue Time Ratio comparison (b) Cost comparison

FIGURE 1.11
Month by Month performance

of pre-submitted VM images or upload their own VM images to EC2. The EC2 service utilises the Amazon Simple Storage Service (S3), which aims at providing users with a globally accessible storage system. S3 stores the user's VM images and, as EC2, applies fees based on the size of the data and the storage time.

Previous work has shown how commercial providers can be used for scientific applications. Deelman *et al.* [4] evaluated the cost of using Amazon EC2 and S3 services to serve the resource requirements of a scientific application. Palankar *et al.*[8] highlighted that users can benefit from mixing Cloud and Grid infrastructure by performing costly data operations on the grid resources while utilising the data availability by the Cloud.

The leading IaaS provider, Amazon, charges based on one hour time block usages. In our scenario we consider unbounded tasks, which can produce a small amount of wasted computation time with the hired resources, detailed evaluation is given in Section 1.6. For example, the policy hires a resource for one hour and the scheduled task on it takes 40 minutes, as the demand is lower the scheduler does not schedule any other task on it. Therefore in the next section, we describe a *clairvoyant* policy that relies on an adaptation of backfilling techniques to give an indication of the impact of wasting this time. We consider the conservative backfilling [7]. In that case the scheduler knows the execution time of tasks beforehand, the scheduler allows a reservation for each tasks when they arrive in the system, and tasks are allowed to jump ahead in the queue if they do not delay the execution of other tasks. Hence, the scheduler can execute smaller task on hired resources that will be next released.

Our work is based on a similar previous work [3] that evaluates differ-

ent strategies for extending the capacity of local clusters with commercial providers. Their strategies aim to schedule reservations for resource requests. A request has a given ready time, deadline, walltime, and a number of resources needed. Here, we consider unbounded tasks that requires a single container to be executed. Tasks are also executed on First-Come-First-Served manner.

Several load sharing mechanisms have been investigated in the distributed systems realm. Iosup *et al.* [6] proposed a matchmaking mechanism for enabling resource sharing across computational Grids. Wang and Morris [14] investigated different strategies for load sharing across computers in a local area network. Surana *et al.* [9] addressed the load balancing in DHT-based P2P networks. Balazinska *et al.* [2] proposed a mechanism for migrating stream processing operators in a federated system.

Market-based resource allocation mechanisms for large-scale distributed systems have been investigated [15]. In this work, we do not explore a market-based mechanism as we rely on utilising resources from a Cloud provider that has cost structures in place. We evaluate the cost effectiveness of provisioning policies in alleviating the increased waiting times of tasks during periods of high demand on the local infrastructure.

1.8 Conclusions

In this chapter, we discuss how a Aneka based hybrid Cloud can handle the sporadic demand on IT infrastructure in enterprises. We also discussed some resource provisioning policies that are used in Aneka to extend the capacity of a local resources by leveraging external resource providers. Once requested, these resources become part of a pool until they are no longer in use and their purchased time block expires. The acquisition and release of resources is driven by specific system load indicators, which differentiate the policies explored.

Using a case study example of Go-Front, we explained how Aneka can help in transparent integration of public Clouds with local infrastructure. It then introduced two policies based on queue length and queue time. We analysed their behaviour from the perspective of the growth threshold, which is in the end responsible of the acquisition of new resources from external providers. The experiments conducted by using the workload trace of October 2006 from Auver Grid show that the best results in term of top queue time and cost are obtained when the growth threshold ranges from 40% to 60% of the maximal value. The behaviour is almost the same for both of the two policies. Being that the average task duration in the October trace is almost 10 hours, we decided to scale down the trace by different factors and explore the behaviour of the policies as the average task size changes. We concluded that

smaller tasks size lead to more wastage when trying to maintain the queue time comparable to average task duration. We then compared the performance of the Queue Time policy with its clairvoyant variant that performs backfilling by relying on the task duration. The experiment show that the improvement in utilisation obtained does not reflect in a corresponding cost improvement. We also introduced a Total Queue Time policy that, in essence combines, the original two policies and studied its performance at different scale factors of the workload. There is only a marginal improvement at the original scale of the trace, where as for smaller task sizes the behaviour is similar.

A comparison with the execution of the same trace without any provisioning capability, clearly demonstrates that the policies introduced are effective in bringing down the Top Queue Time Ratio during peak demands. The cost spent on buying computation time on EC2 nodes to obtain such a reduction in one year can roughly accommodate the purchase of a single new server for the local cluster that definitely does not guarantee the same performance during peak.

Bibliography

[1] AuverGrid. Project website. http://www.auvergrid.fr.

[2] Magdalena Balazinska, Hari Balakrishnan, and Mike Stonebraker. Contract-based load management in federated distributed systems. In *1st Symposium on Networked Systems Design and Implementation (NSDI)*, pages 197–210, San Francisco, USA, March 2004. USENIX Association.

[3] Marcos Dias de Assunção, Alexandre di Costanzo, and Rajkumar Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Proceedings of the 18th International Symposium on High Performance Distributed Computing (HPDC)*, pages 141–150, 2009.

[4] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the Cloud: The montage example. In *2008 ACM/IEEE Conference on Supercomputing (SC 2008)*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[5] 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*, 2008.

[6] Alexandru Iosup, Dick H. J. Epema, Todd Tannenbaum, Matthew Farrellee, and Miron Livny. Inter-operating Grids through delegated matchmaking. In *2007 ACM/IEEE Conference on Supercomputing (SC 2007)*, pages 1–12, New York, USA, November 2007. ACM Press.

[7] 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(6):529–543, 2001.

[8] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon S3 for science Grids: a viable solution? In *International Workshop on Data-aware Distributed Computing (DADC'08) in conjunction with HPDC 2008*, pages 55–64, New York, NY, USA, 2008. ACM.

[9] Sonesh Surana, Brighten Godfrey, Karthik Lakshminarayanan, Richard Karp, and Ion Stoica. Load balancing in dynamic structured peer-to-peer systems. *Performance Evaluation*, 63(3):217–240, 2006.

[10] TOP500 Supercomputers, . Supercomputer's application area share. http://www.top500.org/stats/list/33/apparea (2009).

[11] Jinesh Varia. *Cloud Computing: Principles and Paradigms*, chapter Architecting Applications for the Amazon Cloud. John Wiley & Sons, 2011.

[12] Christian Vecchiola, Xingchen Chu, and Rajkumar Buyya. *High Speed and Large Scale Scientific Computing*, chapter Aneka: A Software Platform for .NET-based Cloud Computing. IOS Press, Amsterdam, Netherlands, 2009.

[13] Christian Vecchiola, Xingchen Chu, Michael Mattess, and Rajkumar Buyya. *Cloud Computing: Principles and Paradigms*, chapter Aneka - Integration of Private and Public Clouds. John Wiley & Sons, 2011.

[14] Yung-Terng Wang and Robert J. T. Morris. Load sharing in distributed systems. *IEEE Transactions on Computers*, C-34(3):204–217, March 1985.

[15] Rich Wolski, James S. Plank, John Brevik, and Todd Bryan. Analyzing market-based resource allocation strategies for the computational Grid. *The International Journal of High Performance Computing Applications*, 15(3):258–281, Fall 2001.