Chapter 1

Introduction

The rapid development of Internet Technologies is opening up new pathways for transforming the way societies and industries work and communicate. Among other things, it has enabled various industries and enterprises to start offloading their applications (henceforth referred to as services) onto third-party infrastructures to achieve cost savings and scalability, get better performance, and deliver on-demand resource provisioning. The introduction of ubiquitous portable computing devices such as laptops, tablets, and smart phones has enabled people to interact with each other using these services at any time and from any location. To satisfy society’s wide-ranging demands, services are becoming increasingly complex software systems consisting of many individual applications that are integrated to provide complex end-user functionalities such as search, social networks, news, and e-commerce. Furthermore, to provide a good quality of service, these integrated applications may require widely varying combinations of resources during their life-span. A key challenge that has emerged as a result is to identify effective means of runtime monitoring and controlling the resource usage of such services, and provisioning them with sufficient computational resources to ensure the desired performance [18, 74, 53, 123, 9, 90].

Computing technologies such as distributed systems, parallel computing, utility computing, grid computing, and virtualization have evolved through a number of phases over time as people have attempted to solve the problem of large-scale shared resource provisioning for services [60, 23, 111, 43, 138, 141, 106, 117, 105, 108, 63]. The emergence of large-scale Internet services coupled with the evolution of these technologies has fueled a trend toward a new resource provisioning paradigm called cloud computing.

The emergence of cloud computing in recent years has been a consequence of the progressive enhancement of these technologies with new features that enable resource provisioning in a flexible, cost effective, and scalable fashion while offering good ease of use. Multiple definitions and concepts of cloud computing have been proposed[12, 95, 142, 156, 54, 11, 102]. According to the NIST [95], cloud computing can be defined as a model that provides large-scale computing
farms based on the utility computing service model \(^1\) to enable convenient on-demand network access to a shared pool of configurable computing resources such as CPU cores, networks, storage, and memory, which can be quickly provisioned and released with minimal management effort. The resources are usually geographically dispersed around the world in datacenters consisting of tens of thousands of commodity machines \([24, 70, 100]\). The number of machines available in this way will probably continue to grow for the foreseeable future because the demand for hosted services and remote resources is increasing steadily \([156, 59, 67]\).

The challenge and complexity of provisioning resources for services on demand are increasing rapidly due to the growing interest in cloud computing among both the general public and industrial actors, and the rapid developments in the size and diversity of cloud computing resources and the services hosted on them \([24, 70, 100, 156, 109, 77, 148]\). At the large scales that are becoming increasingly common, it is necessary to account for the fact that there will always be a significant number of servers and network components that have failed at any given time. Further complexity is introduced by the heterogeneity of hosted services and the unpredictability of their workloads. Due to this heterogeneity and the sheer scale of cloud computing systems, there is an increasingly strong expectation that services should be self-configuring and self-managing. However, as the range of permissible configurations grows it is becoming increasingly difficult to achieve this goal.

The cloud infrastructure as a whole should not be simply regarded as heterogeneous collections of co-located commodity hardware that have been wired up together. Rather, it should be seen as a set of commodity machines that have been aggregated seamlessly and transparently into a single computing unit \([24, 72, 16, 71, 144, 122, 130, 35]\). Such an aggregation of commodity-class machines into a single computing unit provides an illusion of an operating system running on top of an infrastructure of tens of thousands of servers. This vision for large-scale systems can only be realized through advancements in the software management layer that abstracts the infrastructure \([146, 72, 69, 121, 7, 34]\).

The management layer is responsible for many tasks such as the deployment of services to hardware resources, resource provisioning, scheduling, enforcement of quotas, monitoring, load balancing, resource usage collection, and handling component failures. These tasks must all be managed in a way that imposes a minimal overhead on the underlying system. The management layer is also responsible for ensuring that the system satisfies all of the hosted services’ non-functional requirements relating to performance, reliability, scalability, availability, latency, and fault tolerance.

The task of the management layer for a large-scale distributed system has become more complex than the elements being managed. Consequently, as

\(^1\)The utility computing service is a service provisioning model in which a service provider makes computing resources (e.g. processing power, storage, and high level services) available to the customer as needed, and charges the customer on a metered basis as is commonly done for water and electricity.
the demand for intricate distributed services has grown, the development of management tools for such systems has become a complicated problem in its own right [112, 116]. Due to the sheer complexity of the managed elements, it is imperative for the management system to be autonomous. That is to say, the management system should continually adapt the whole system to changes in environmental variables such as the workload, hardware availability, and software failure by automatically adjusting the decision variables [81, 33, 13] without any human intervention. Specifically, such solutions should have capabilities such as self-configuration, self-optimization, self-adaptation and self-healing that allow the system to control its managed elements and facilitate continuous functioning in the face of unpredictable changes in the infrastructure and hosted services.

This thesis focuses on the design and implementation of efficient algorithms, models and techniques for the autonomous monitoring, control, and provisioning of the diverse resources required to meet the demands of services and account for their resource usage. The aim is to develop effective tools for (i) enforcing quotas, (ii) provisioning optimal amounts of physical resources to meet service performance requirements, and (iii) collecting and aggregating service resource usage data for accounting and billing purposes in a timely fashion at runtime without any service interruption.

Quota management mechanisms are essential for controlling distributed shared resources to ensure that services do not exceed their allocated or paid-for budgets of credit, CPU hours, CPUs, RAM, storage capacity, and so on. Quota management tools for services must have mechanisms that both regulate resource usage and achieve an efficient and fair distribution of quotas among services. Cloud-wide monitoring and control of quotas is needed to achieve these aims and avoid the over- or under-provisioning of resources. Papers I and II focus on managing quotas for services running across distributed nodes.

Quotas that apply over multiple nodes must be mapped onto physical resources on the basis of services’ performance requirements. It is important to be able to continuously adjust the physical resources allocated to services at runtime in order to achieve guaranteed performance levels at all times in an environment where frequent and unpredictable changes are the norm rather than the exception. However, this is very challenging because of multi-faceted issues such as the dynamic nature of cloud environments; the need for supporting heterogeneous services with different performance requirements; the unpredictable nature of services’ workloads; the complexity and differences among Key Performance Indicators (KPIs) used by each service, and the non-triviality of mapping performance measurements into resources. Papers IV–VIII present models and techniques for addressing these issues and managing temporary resource shortages due to e.g. flash crowds or hardware failure.

Furthermore, run time monitoring data should be collected for each service

---

2The term quota as used in this thesis has the same meaning as the phrase resource allocation as used in paper I; quota is used throughout this thesis to avoid the risk of confusion with resource provisioning.
and aggregated in order to maintain a single global state of the system that facilitates management activities such as accounting and billing, scheduling, load balancing, and network analysis. Such aggregation becomes extremely complex when dealing with systems spread over geographically dispersed locations, which means that the management task itself may consume significant computing and network resources unless done with care. Paper III focuses on collecting data across multiple nodes and establishing mechanisms to ensure consistency and synchronization in order to facilitate accounting and billing operations.

The thesis is organized as follows. Chapter 2 introduces the problems that the thesis aims to address. Chapter 3 reviews established methods for solving the quota and usage management problems. Chapter 4 briefly presents state-of-the-art techniques for resource provisioning. Chapter 5 summarizes the new contributions presented in this thesis. Finally, 8 papers produced in the course of the thesis work are appended.
Chapter 2

Runtime Resource Management and Control in Cloud Computing

Resource management and control is perhaps the most important task that must be performed to deliver a reliable and resource-efficient cloud computing system [52, 72, 150]. Each service’s resource consumption must be continuously tracked at runtime to determine whether services are meeting their target Service Level Objectives (SLOs). Cloud infrastructures consist of many tens of thousands of units of commodity hardware with thousands of services deployed, all of which have different requirements and demands. Effectively running such an infrastructure is extremely complicated and challenging because there are so many variables that must be considered such as unpredictable service demand changes, hardware failures, and interference between collocated services. Logically, different approaches and methods that can reflect the different variables must be employed in order to allocate and control the available resources in the best possible way.

For the purpose of our discussion we assume that these resources can be divided into virtual and physical resources, a division that can be likened to the distinction between intangible and tangible assets in business and economics [26, 10]. Virtual resources are abstract resources such as prepaid credits, quotas (for CPU, storage, etc.), and accounting records while physical resources are resources that have actual physical existence such as CPU cores, RAM, and Storage. Accordingly, the control tasks are divided into quota and resource usage control and resource provisioning. Quota and resource usage control entails runtime tracking and controlling, collecting and aggregating virtual resources (i.e. quotas and resource usage) for each service, while the resource provisioning task is responsible for autonomic runtime prediction and allotment of optimal physical resources to services to meet their requirements.
Note that from this point onwards, when we talk about resources in the context of quotas and resource usage, we are referring to virtual resources. Conversely, in discussions about resource provisioning, resources means physical resources.

The following sections discuss the issues associated with controlling these two resource types in detail.

2.1 Quota and Resource Usage Control

Large-scale services often need sophisticated and scalable quota management and resource usage monitoring support. It is important to collect resource consumption statistics across nodes, clusters, or datacenters to facilitate network analysis and to support management decisions relating to accounting and billing, scheduling, and load balancing among other things. For example, resource usage data may need to be collected from several locations in order to generate a single bill for a customer’s service consumption across the entire infrastructure.

Figure 1 presents a model of a cloud computing environment. We will use this model throughout this section when we talk about quota and resource usage control. The cloud customer deploys and runs services on a cloud infrastructure. A cloud customer may be a service provider, who leases resources offered by an infrastructure provider to host services that will be used by end-users or other service providers. The services are customers’ applications, which are deployed across multiple clusters where they consume resources and impose resource demands on the cloud infrastructure. The cloud infrastructure consists of geographically distributed clusters, each of which is composed of thousands of commodity servers connected to a low-latency Local-Area Network (LAN) [84].
The infrastructure provider is responsible for managing the clusters’ physical and virtual resources and controlling the resource usage of each individual service running across multiple clusters. The focus here is on how to manage quota and resource usage for services running across clusters.

*Quotas* and *resource usage* control mechanisms for services running across clusters profoundly influence the performance of the entire system, the provider’s profitability, and the level of customer satisfaction. For example, to maximize customer satisfaction, the service provider should make sure to avoid over-committing the infrastructure’s resources [112, 150]. Moreover, resources consumed by different services should be monitored and this monitoring data should be collected efficiently in a way that interferes minimally with the services’ performance [48, 68, 61, 46, 118, 47]. Ideally, resource usage collection and quota management mechanisms should not impose significant constraints on either the customer or the cloud provider.

*Quota* control or enforcement systems are tools for monitoring, redistributing, and assigning shared resources (such as prepaid credits and quotas) to individual instances of a service running across multiple nodes based on each instance’s demand, subject to the constraint that the sum of shares across instances must not exceed the aggregated global availability of these resources. Quota control mechanisms should guarantee that each individual instance’s demands are met without over-committing resources. In other words, the management mechanisms should ensure that the aggregate resource consumption does not exceed the quota limit while simultaneously ensuring that instances running on particular nodes are not starved when the total consumption is below the quota limit. To this end, quota allocation mechanisms should know the current allocation across nodes and be able to use this information to intelligently apply algorithms to better distribute resources to instances according to their runtime requirements [115, 149, 154]. The goal is to avoid both under- and over-utilization of quotas.

On the other hand, *resource usage* management systems are tools for collecting and aggregating data about resources consumed by services across clusters in order to establish a single consistent global view for purposes such as accounting and billing [47, 48, 153, 152, 6], creating traces [132, 58, 133], and to provide input for provisioning [36, 8, 64, 83, 129, 134, 143, 76, 45, 93]. Resource consumption data should be monitored continuously and must be collected and synchronized to maintain a single consistent view of the data generated across different clusters. However, any such synchronization mechanism must make a trade-off between the consistency of the data and the performance of the infrastructure: more frequent synchronization increases data consistency but reduces performance.

The following sections discuss quota and resource usage management in more detail.
2.1.1 Quota Control

Service owners and/or infrastructure providers may want to cap (put limits on) cloud resource quotas such as the total credit, storage quota, total number of CPUs, total CPU hours, or number of VMs, IP addresses, or network connections. To do this, the provider or owner must have a mechanism for allocating and enforcing limits across services running on multiple nodes [96, 79]. This is similar to the way that prepaid telecommunication services work [2, 1]: customers pay in advance and use the service afterwards. The mechanism is expected to provide the service to customers immediately as long as the paid-for budget is not exhausted.

Managing quota limits in large distributed systems is difficult because it requires the maintenance of a consistent picture of total usage while resources are being consumed concurrently at several locations. Quota management is essential for ensuring that services do not exceed their allocated (or paid-for) budget. It can also be used to enforce global allocation limits in order to solve problems such as spurious services that flood and overburden the system with dummy tasks, denying access to other services, and malicious services that launch Distributed Denial of Service (DDoS) attacks.

The resource consumption of services running on tens of thousands of nodes in many clusters must be properly monitored. Furthermore, quotas should be distributed across nodes or clusters optimally, with the objective of minimizing their associated overhead while meeting service demands.

2.1.2 Resource Usage Management

Resource usage management involves monitoring, collecting, and aggregating information on resource consumption in order to facilitate decision making in cloud environments [32, 128, 99, 126, 38]. It requires a transparent and efficient management mechanism that can be used to produce a consistent cloud-wide view of the data. The degree to which resource usage must be managed, i.e. the necessary update frequency and the desired granularity of the monitored statistical information, will vary according to the management task at hand.

Each service may use resources from multiple clusters and can generate a variety of runtime scenarios that, if not tracked and responded to, can make usage-based reporting and billing impossible. Resource usage must be monitored, synchronized, and aggregated to perform billing operations. Thus, for cloud services to be commercialized using a pay-as-you-go model, the cloud must allow runtime usage across clusters to be accurately measured, collected and synchronized.

Therefore, an accounting and billing system capable of monitoring, collecting, and processing usage data and metrics should be incorporated into the cloud architecture to support the business model of cloud providers. Once services are deployed to the cloud, it is critical to monitor their functions and track their resource usage on an ongoing basis in order to determine how much service
owners are to be charged. Timely and consistent collection and distribution of accounting data is not only important for the infrastructure provider’s billing operations: service owners may also need to bill their customers on the basis of resource consumption. Similar situations occur in the supply chains of manufacturing industries [97, 140], where a faulty material or a delay in the supply of a component will affect all of the downstream processes in the chain. Likewise, a delay or a failure in infrastructure monitoring will affect all members of the ecosystem.

2.2 Resource Provisioning

A cloud is a complex system with a very large number of shared resources whose performance and capabilities may be limited by the sometimes unpredictable behavior of the infrastructure and the services running on top. At the same time, services hosted in the cloud have become indispensable for many business and personal purposes, making the performance of these services a key issue [66, 39, 75, 73, 41, 135, 157]. Resource provisioning techniques help to determine the optimal amount of physical resources required to satisfy service demands. Today, cloud infrastructure providers either do not offer any performance guarantee or prefer coarse-grained and static resource provisioning with fixed sizes, resulting in inefficient resource utilization and Service Level Agreement (SLA) violations. Infrastructure providers will need to provide better and more stringent performance guarantees with fine-grained resource provisioning techniques to attract businesses to move their core services into the cloud.

Resource provisioning has a direct influence on the performance of services in the short-term and the survival of the infrastructure provider in the long-term. It is important to have autonomic techniques that continuously adjust services’ resources at runtime in order to provide performance guarantees at all times in an environment where frequent and unpredictable changes are the norm rather than the exception. Indeed, autonomic resource provisioning technique is a necessity rather than a luxury due to the scale and complexity of cloud infrastructures, the huge number of services deployed on them, and the unpredictability of runtime changes in services’ capacity demands (which may produce sudden demand spikes known as flash crowds) and operating environments, i.e. hardware failures [83, 107, 81, 76, 65, 44].

Ideally, the infrastructure provider should have a resource provisioning mechanism that maximizes resource utilization while using dynamic and fine-grained resource allocation to guarantee that services’ performance requirements are met. Thus, the task of resource provisioning is to allocate services enough resources to meet their performance requirements, but no more. Unfortunately, this is very difficult to do in practice because of the dynamic nature of cloud environments, the need for supporting heterogeneous services with different performance requirements, the unpredictable nature of cloud workloads, the complexity of and differences between the KPIs used by each service, and the
non-triviality of mapping performance measurements into resources [49, 17, 25, 135, 157].

Given a particular desired service performance, the goal is to determine the optimal amount of hardware resources required based on the services’ runtime behavior. Specifically, service owners should be able to specify their service resource requirements in terms of their preferred KPIs (e.g. response time, throughput, CPU and memory utilization, etc) and have the resource provisioning techniques translate those into optimal resource allocations that satisfy the SLO for each service.
Chapter 3

Resource Usage and Quota Management

Infrastructure providers find themselves in an era where customers expect their services to be delivered in a way that meets certain requirements with respect to performance, capacity, cost and geographical distribution. Recently, there has been considerable interest in using software driven infrastructure management [57, 101, 113, 78] to manage cloud infrastructures and seamlessly deliver resources to customers’ services. The idea behind software driven infrastructure management is to have a software suite that provides a comprehensive abstraction of a complete large-scale distributed infrastructure similar to the way that an operating system abstracts the different components of a computer. However, traditional management mechanisms, tools and techniques cannot meet the requirements of software driven infrastructures because of their limited scaling [68, 155, 113, 78].

Software driven management of infrastructures is rapidly becoming a new target for IT enablement, but this concept will only be fully realized through the development of improved management mechanisms.

As discussed in Chapter 2, successful quota and resource usage management of services at the cloud scale requires a rich set of global information and global control for each individual service running across the infrastructure’s nodes. As the number of nodes and the services’ size increase, the overhead of collecting, analyzing, and acting upon the associated data grows. The basic features expected from the chosen approach are:

- **Accuracy**: Resources should be monitored, collected, aggregated or controlled accurately without being over- or under-stated.

- **Scalable**: The mechanism should remain functional and viable as the number of services and nodes increases.

- **Fault tolerance**: Partial failure of the mechanism should not bring the
whole infrastructure to a halt.

- **Less overhead**: Resources consumed by services should be monitored and collected in a way that interferes minimally with hosted services.

The management layer can be designed in centralized or decentralized fashion as described below. Each approach has its own advantages and disadvantages and is affected by a number of factors such as the size of the managed entities (measured in terms of, e.g., the number of nodes or services) and the degree of geographic dispersion.

### 3.1 Centralized Management

In a centralized management system, a centralized manager keeps track of all available resources, collects usage from different managed entities and makes decisions. In general the management functions and decision-making are concentrated at a central component. All managed entities send messages requesting resources and reporting resource usage to the manager. The manager is responsible for allocating resources to services and storing and processing resource usage data. With such centralized designs, few decisions need be made about the allocations and usage of resources.

The majority of cloud computing platforms rely on centralized architectures combined with a hierarchical system of control [19, 52, 67]. The main advantage of this scheme is simplicity. Moreover, it eases consistency concerns and avoids conflicting decisions by providing central control over the whole system.

However, centralized management presents the well-known problems of single point of failure and single point of congestion. Moreover, the centralized manager is burdened with multiple decisions for all managed entities and a lot of communication needs to take place from different nodes hindering the performance of the whole system. Thus, as the system size increases, the centralized manager gets heavily loaded and becomes a bottleneck. It may also limit concurrency. In general, it suffers from poor availability, lack of fault tolerance, and limited capacity for scaling.

### 3.2 Decentralized Management

In recent years, there has been an increased and wide interest in decentralized management techniques and cooperative distributed decision making [125, 124, 94]. The central notion of decentralized management is to pursue full local autonomy while cooperating (by communicating) to achieve a global goal. The ideal is that individual management units are able to acquire information about the state of the entire system by communicating their local information to all others. However, fine-grained information about the entire system is generally not available due to the vast amount of local information that may be
generated; if all of this information were transmitted in its entirety, it would cause significant network congestion. In addition, the information received may not be up-to-date due to delays.

Decentralized management disperses the decision-making process over multiple management components so as to alleviate the problems encountered with a centralized approach. Managers at specific locations make key decisions relating to their sphere of responsibility and cooperate with peers as necessary.

In such an architecture, all of the peers involved in a management task play similar roles, interacting cooperatively without any one having a distinct role. The aim of this scheme is that each management unit performs its responsibilities independently without being hampered by the state of others, and only interacts with others when it is not possible to achieve the global management goal locally. Decentralized management can also abstract the differences between managed entities in different locations without limiting their capabilities. It is thus a viable method for achieving a stable growth pattern in an era of rapid technological development and investment, which allows each peer to have different internal management policies and mechanisms.

The main characteristics of a decentralized approach are:

- **Increased Availability**: Availability refers to the accessibility of a system in the presence of failure. For a centralized design, if the central management or the cluster where the management is located fails, the service becomes unavailable for other clusters as well. In contrast, with decentralized management the failure of one or more clusters does not prevent the remaining clusters from providing the service among themselves.

- **Fault tolerance**: Fault tolerance is the ability to function in the presence of component failures without performing incorrect actions. With decentralized management, the failure of one or more clusters can be tolerated as it only affects those services running on the failed clusters.

- **Enhanced performance**: Since services are running at multiple clusters, requests for different services do not have to line up at one management component; instead they can line up at multiple management components located on different clusters to reduce their waiting time.

- **Better Scalability**: In general, a decentralized management scales well as the sizes of services and clusters grow.

- **Greater Autonomy**: The decentralization approach gives individual clusters autonomy in making their own decisions. A decentralized approach assumes that each entity is autonomous and self-controlled, making its management decisions based on its own policies. Local management units need only coordinate with other clusters when local information is not sufficient for making decisions.

There is a growing trend towards using decentralized management approaches to support management demands in different computing domains because the
current centralized model is increasingly failing to achieve its goals. For example, decentralized techniques have been used in quota management across nodes inside a LAN [110, 82, 22, 21, 79, 110] as well as for network management [20, 96, 114, 149, 80, 42, 5]. The goal of this thesis is to apply decentralized techniques to the quotas and resource usage of services running on multiple clusters.

Figure 2: A decentralized management approach with distributed management entities.

Figure 2 shows the architecture of a decentralized management system. Management components interact with all or part of other management components in different situations. The arrows in the figure indicate interactions among management units. Some management units can work alone without interacting with other peers as long as their decisions can be made entirely locally. A hybrid management approach can also be used [29, 51, 56] by incorporating some features of both the centralized and the decentralized approaches. Hybrid approaches generally rely on a central unit, the coordinator, that coordinates the actions of other units. The coordinator is responsible for performing global operations such as processing requests from other management units and making global decisions. The other management units control local resources and report status information to the coordinator. Moreover, each management unit can make local decisions and may contact the coordinator only when unable to do so. This approach may suffer from similar issues as centralized approaches due to the reliance on a single coordinator.
Chapter 4

Resource Provisioning

Virtualization technologies are among the key enablers of cloud computing. They simplify some resource management tasks such as the efficient utilization and sharing of resources by consolidating diverse enterprise services, each wrapped inside a Virtual Machine (VM) on a single Physical Machine (PM). These technologies enable resources such as CPU cycles, memory, secondary storage, and I/O and communication bandwidth, to be added or removed at runtime from one service to another on demand.

Nonetheless, current virtualization technologies are inadequate for determining resource demands and assuring performance guarantees for hosted services. Moreover, hosted services are likely to manifest emergent behaviors that lead to unhealthy resource contentions and undesirable performance interference as a result of co-location [151, 30]. For example, the performance of one service may be affected by demand changes in other co-located services. Moreover, the infrastructure may be overloaded due to sudden changes in services’ capacity demand – flash crowds – and operating environments – hardware failures [83, 107, 81, 76, 65, 44].

Consequently, most existing cloud services are targeted at consumers who have low Quality of Service (QoS) expectations. However, due to its potential economic benefits, increasing numbers of enterprises wish to exploit the advantages of the cloud computing model such as pay-per-use pricing and rapid elasticity. To retain existing customers and attract new ones, existing virtualization technologies must be augmented with techniques that capture the runtime behavior of hosted services and determine the optimal resource allocation pattern for meeting each service’s SLO. Specifically, mathematical and statistical models and techniques are needed to correlate resources and runtime performance behaviors in response to external changes such as changes in workloads or the infrastructure.

For the purpose of discussion, resource provisioning can be categorized into Capacity-based and Performance-based provisioning. Capacity-based approaches use resource utilization (e.g., CPU and memory utilization) when making re-
source provisioning decisions while performance-based provisioning uses metrics extracted from services (e.g. response time, throughput, number of requests). The sections below provide brief discussions about the two resource provisioning approaches as well as the management of capacity shortages.

4.1 Capacity-Based Provisioning

Cloud providers use resource utilization to guide their resource management [91, 145, 131]: resource utilization data are used to predict services’ resource requirements. Efficient resource provisioning that accounts for resource utilization reduces operating costs, for example by allowing providers to consolidate VMs onto a small group of machines so that other machines can be shut down to reduce energy consumption [139, 104, 37, 50]. Consolidation is an inherently effective method for increasing resource utilization, and in turn reduces energy consumption. It can also free up resources for use by other services. However, utilization-based decisions may lead to SLA violations as they are oblivious to service performance.

4.2 Performance-Based Provisioning

A number of studies have suggested that end-users are sensitive to performance changes. For example, Amazon loses 1% of its sales for every 100ms of latency [3] while an extra half second’s delay in search page generation reduces Google’s traffic by 20% [62]. A broker could lose $4 million in revenues per millisecond if their service was 5 milliseconds behind the competition [4]. In general, several studies have shown that end-users will abandon an e-commerce service for a competitor if its response time is above 4 seconds [103], thus incurring long-term revenue loss. In the same manner, video streaming services can also expect users to abandon their services if the expected level of throughput drops.

All these studies demonstrate the importance of considering performance during resource provisioning. Capacity-based approaches are inadequate to ensure performance because they are oblivious to the observed performance of services. However, performance-based resource provisioning is challenging due to the non-trivial relationship between performance and capacity, and the unpredictable nature of incoming workloads.

4.3 Capacity Shortage Management

A challenging problem in cloud infrastructure management is how to deal with short- or long-term capacity shortage management, i.e. situations in which the aggregate demand for resources exceeds the resources available in the infrastructure. Several techniques such as elasticity, replication, migration, and load balancing have been suggested to mitigate this issue as long as there
is spare capacity in other hosts [15, 27, 129]. However, it is economically infeasible to reserve enough spare capacity since unexpected spikes can increase resource demand five-fold [31]. Moreover, some services may not tolerate any disruption at all. Because of this, most cloud services have targeted consumers with low performance expectations as noted above, and enterprises remain hesitant to move core business services into the cloud. Enterprises will require guaranteed performance under all conditions, expressed in terms of appropriate Key Performance Indicator (KPI) metrics (e.g. response time or throughput), in order to be confident in moving key services into the cloud.

Service differentiation schemes that decide which services to degrade can be used to address short-term capacity shortage problems [127, 119, 14, 92, 136, 98, 28, 147, 40, 158, 107]. Specifically, during service differentiation decisions, services with stringent performance needs are given higher priority than services with relatively lower performance requirements. Each service can be placed into different SLA classes (e.g., gold, silver, and bronze), signifying their relative importance. Each class is assigned a penalty weight that stipulates the relative importance of the corresponding services. This makes it possible to preferentially maintain the SLOs of higher class services by shifting resources from less performance-sensitive ones during periods of infrastructure overload.

Fig. 3 shows a logical view of a high-level architecture for an autonomic service and resource manager that would allocate the optimal resources required for each service deployed in the cloud. Note that the realization of the manager can be centralized or distributed. In general, the manager can make decisions concerning vertical and horizontal scaling of VMs, VM migration, load balancing, and service priority so as to meet service’s SLO requirements and ensure efficient resource utilization. The focus in this thesis is on performance-based and fine-grained resource provisioning by resizing VMs (i.e. vertical scaling) to ensure that services’ performance requirements are met. In the event that the available resources are temporarily insufficient to satisfy the demand, service differentiation is applied in order to guarantee the performance of critical services.

The following features would be expected of an autonomous resource manager:

1. **Autonomic resource provisioning.** A self-adaptive resource provisioning that dynamically adjusts resources to services based on their workload dynamics and relative importance without requiring human intervention should be employed.

2. **Timely detection of changes.** The changes in the workload behavior as well as in the infrastructure should be detected immediately in order to react to them accordingly.

3. **Fast computation.** Fast computation is needed in order to react immediately to changes in the services’ performance behaviors.
4. **Guaranteed performance.** The resources allocated should be exactly sufficient to satisfy the demand. That is to say, services should be provided with enough capacity to meet their performance targets, but no more. As a result, capacity over-provisioning or under-provisioning should be avoided under normal conditions.

5. **Service prioritization.** Capacity shortages may happen at any time in complex systems such as cloud infrastructures. Thus, when the capacity demand exceeds the supply, resources should be allocated based on service priorities using their performance as an indicator. Such priority schemes should be independent of services’ KPIs and should ensure that the performances of higher priority services is less affected than their low priority counterparts.
Chapter 5

Summary of Contributions

This section presents the contributions of the thesis. Fig. 4 provides a conceptual view of these contributions. At the top are infrastructure-wide quota enforcement mechanisms which ensure that hosted services run without issue as long as the global quota is available. At the bottom, a non-intrusive usage accounting mechanism collects, aggregates and synchronizes the resource usage of services across the infrastructure to facilitate timely billing operations. The autonomic resource manager, deployed at each node, implements various techniques to provision resources according to services’ demands.

Papers I, II and III present highly scalable solutions for managing quota enforcement and usage accounting efficiently and consistently. The proposed solutions scale with the size of the infrastructure as well as the services deployed on top of it. The proposed solutions are also robust since instance failures are localized and the system continues to operate and deliver useful service in the event of a local failure.

Papers IV–VIII present models and techniques that map service performance requirements onto physical resources. The proposed solutions continuously adjust the allocation of resources to meet the services’ performance requirements irrespective of workload variation. Moreover, the proposed mechanism guarantees to meet the performance requirements of critical services during capacity shortages by shifting resources from less critical services.

5.1 Distributed Quota Enforcement

5.1.1 Paper I

Paper I [85] focuses on how to manage quota for services running in geographically distributed clusters. The paper presents a fully distributed quota management scheme for tracking and distributing quota shares across distributed clusters. The solution monitors resource consumption by services that are spread over a number of clusters with the goal of triggering global polls
only when the allocated balance in a cluster decreases below a threshold and allocations are reassigned in a way that avoids further immediate global polls.

Unlike a single coordinator or predefined coordinator, the paper proposes a distributed scheme that selects a per-event coordinator when needed. For each allocation event, the coordinator is selected and the clusters currently running that service form a multi-cast group to perform the allocation algorithm.

It achieves scalability by minimizing global message exchanges, increases performance by distributing requests, and improves availability by avoiding a single point of failure. A range of simulations were performed and the communication cost of the fully distributed allocation approach is compared to that for a reference centralized resource manager algorithm, showing that the distributed approach is more efficient and effective.

5.1.2 Paper II

Paper II [89] extends the work presented in Paper I by adding support for multiple parallel decisions by arranging nodes using a spanning tree. Querying every node during quota redistribution decision causes latency and communication overheads as the number of nodes and the distances between them increase. Here, we address the latency and communication overhead problems by proposing a tree-based protocol for enforcing quotas in clouds. In our protocol, all nodes running a service maintain a local quota allocation so that permissions can be given immediately provided that the local allocation is not exhausted. When local quota allocations are exhausted, nodes request extra quota from nearby neighbors progressively instead of polling all nodes.
5.2 Distributed Resource Usage Management

5.2.1 Paper III

Paper III [88] presents monitoring, data-collection, and synchronization mechanisms in multi-cluster environments so as to facilitate accounting and billing operations.

With today’s large-scale, geographically distributed clusters, running customers’ services across multiple clusters is a possibility. As a result services generate huge amounts of accounting records that are dispersed throughout all clusters. Available accounting systems that are centralized cannot efficiently manage these huge amounts of usage records generated across multiple clusters. To cope with this problem, the paper proposes a new approach for monitoring, collecting and synchronizing accounting records in a decentralized fashion that provides a desired level of scalability.

A distributed mechanism is employed for collecting and synchronizing accounting records across a number of geographically distributed clusters as well as serving accounting requests from any of the clusters in a consistent manner. The mechanism collects and merges accounting records generated from all clusters and maintains the intrinsic requirements of accounting.

5.3 Performance-based provisioning

Capacity-based resource provisioning systems where resources are allocated based on utilization thresholds are oblivious to the observed performance of services and cannot be readily used to meet the performance needs of services expressed using KPIs such as response time or throughput. Papers IV–VI address this issue.

5.3.1 Paper IV

Paper IV [86] presents two generic performance models– the previously proposed queue length model and the novel inverse model– for mean response time that map performance to capacity in order to provide performance guarantees for interactive services deployed in the cloud. We carried out an extensive set of experiments using RUBiS, RUBBoS, and Olio–three widely used interactive cloud benchmarking applications– with varying workload mixes over time under both closed- and open-system models [120]. We also varied the target response time of each application to see how the models behave. The results demonstrate that the two models are stable for higher response time targets. However, our inverse model exhibited greater stability than the queue length model for lower targets.
5.3.2 Paper V

Paper V [86] presents two novel tail response time performance models-- the queue length tail model and the inverse tail model-- for predicting the capacity required to guarantee tail response times for interactive services deployed in the cloud. The two models were able to allocate just the right amount of capacity to meet tail response time targets expressed in percentiles (e.g. 95% or 99%) while avoiding substantial capacity over- or under-provisioning.

Both models were evaluated in an extensive set of experiments using RUBiS, RUBBoS, and Olio with real workloads and synthetic workload mixes that varied over time under both closed- and open-system model. We also varied the target response times of each application to see how this affected the models’ behavior. Our results demonstrate that both tail response time models are stable under both more predictable and unpredictable real workloads and synthetic workloads generated using open- and closed-system models [120]. However, the inverse tail response time model is more stable than the queue length tail response time model.

5.3.3 Paper VI

In paper VI [55], we proposed an autonomic resource controller consisting of three sub-controllers-- a fuzzy controller, a CPU controller, and a memory controller-- to allocate the right amount of CPU and memory in order to meet mean response time targets for interactive services. The fuzzy controller acts as a coordinator, ensuring that the control actions of the cpu and memory controllers complement each other in order to fulfill the service’s performance requirements, which were expressed in terms of mean response times. The CPU controller and memory controller allocate the right amount of CPU and memory, respectively, using the inputs provided by the fuzzy controller. In general, the proposed fuzzy control approach can be used as a coordination technique for distributed controllers. We evaluated the proposed solution with RUBiS, RUBBoS, and Olio in a virtualized environment using Xen Hypervisor. Different experiments were conducted under workload traces generated based on open and closed system models. The results show that the proposed coordination solution was able to maintain the target response time with while achieving high resource utilization.

5.4 Performance-based service differentiation

5.4.1 Paper VII

One of the many advantages of cloud computing is the ability to consolidate multiple services into a limited number of servers so as to achieve high datacenter utilization. However, maintaining high utilization while meeting SLOs is difficult, as a datacenter may become overloaded, reducing the performance of the hosted
services. Service differentiation has been proposed as a way of controlling which services get degraded. In paper VII [87], we propose performance-based service differentiation where capacity is distributed among services based on their observed performance and sensitivity to performance degradation. Specifically, when enough capacity is available, each service is automatically allocated the optimal amount of capacity to meet its target performance, expressed in terms of either response time or throughput. For cases when the available capacity is not sufficient, we propose two service differentiation schemes that dynamically decide which services to degrade and to what extent. We carried out an extensive set of experiments using different services—interactive as well as non-interactive—in which the services’ workload mixes were varied over time. The results demonstrate that our solution precisely provides guaranteed performance or service differentiation depending on available capacity.

5.4.2 Paper VIII

Cloud storage is increasingly being adopted by users as simplified storage systems become available. These systems are mostly presented as Object Storage Systems (OSSs), hiding issues such as redundancy from users. As new industries are considering adopting clouds for storage, OSSs must evolve to support new needs. Among the most challenging is assuring guaranteed performance. In paper VIII [137], we present Controllable Trade-offs (CTO), an OSS-agnostic solution for providing performance guarantees. CTO presents itself as a thin layer that mediates requests between the user and the OSS. For generic support, performance is controlled by tuning the rejection probability based on priorities associated to each customer. Results show that CTO may reduce penalties by a factor of 3.23 on average and by a factor of up to 68 when the load is high.
Bibliography


[3] Amazon found every 100ms of latency cost them 1 Available online: http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/, Visited 2015-04-06.


