CHAPTER 1
INTRODUCTION AND MOTIVATION

1.1 Introduction: Internet Applications and Hosting Platforms

1.1.1 Internet applications

An Internet application is an application delivered to users from a server over the Internet. A popular class of Internet applications consists of Web applications such as Web-mail, online retail sales, online auctions, wikis, discussion boards, Web-logs etc. Web applications are popular due to the ubiquity of the Web browser as a client, sometimes called a thin client. The ability to update and maintain Web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity. Not all Internet applications are Web based, for example some streaming media servers [105] or game servers [45]. During the past decade we have increasingly come to rely on these applications to conduct both our personal and business affairs. We use the terms Internet application and Internet service interchangeably in this thesis.

1.1.2 Data Centers

A data center is a facility used for housing a large amount of electronic equipment, typically computers and communications equipment. As the name implies, a data center is usually maintained by an organization for the purpose of handling the data necessary for its operations. A bank for example may have a data center, where all its customers’ account information is maintained and transactions involving this data are carried out. Practically

---

1 Notice that our focus is exclusively on applications based on the client-server model. We do not consider the recently popular peer-to-peer applications [47, 80] in this work.
every company mid-sized and upwards has some kind of data center, and large companies often have dozens of data centers. Most large cities have many purpose-built data center buildings built to provide data center space in secure locations close to telecommunications services.

**Physical Layout:** A data center can occupy one room of a building, one or more floors, or up to the whole building. Most of the equipment is often in the form of 1U servers (so-called "pizza boxes") racked up in 19 inch rack cabinets, which are usually placed in single rows forming corridors between them. This allows people access to the front and rear of each cabinet. Some equipment such as mainframe computers and storage devices are often as big as the racks themselves, and are placed alongside them.

- Air conditioning is used to keep the room cool, generally around 17 degrees Celsius (about 63 degrees Fahrenheit). This is crucial since electronic equipment in a confined space generates much excess heat, and tends to malfunction if cooling is not handled.

- Backup power is often available. This can include one or more uninterrupted power supplies and diesel generators located close by.

- To prevent single points of failure, all elements of the electrical systems, including backup system, are typically fully duplicated, and critical servers are connected to both the ”A-side” and ”B-side” power feeds.

- Data centers typically have raised flooring made up of 2 foot (600mm) removable square tiles. These provide a plenum for air to circulate below the floor, as part of the air conditioning system, as well as providing space for power cabling. Data cabling is typically routed through overhead cable trays in modern data centers.

- Data centers often have elaborate fire prevention and fire extinguishing systems.
Physical security also plays a large role with data centers. Physical access to the site is usually restricted to selected personnel. Video camera surveillance and permanent security guards are almost always present if the data center is large or contains sensitive information on any of the systems within.

**Network Infrastructure:** Communications in data centers today are most often based on networks running the IP protocol suite. Data centers contain a set of routers and switches that transport traffic between the servers and to the outside world. Some of the servers at the data center are used for running the basic Internet and Intranet services needed by internal users in the organization: email servers, proxy servers, DNS servers, etc. Network security elements are also usually deployed: firewalls, VPN gateways, Intrusion detection systems, etc. Also common are monitoring systems for the network and some of the applications.

### 1.1.3 Hosting Platforms

Due to the prevalence of Internet applications, data centers that host them have become an important and attractive business. We refer to such data centers as *hosting platforms*. To make an application available to the Internet community, it needs to be hosted on one or more servers. For example, a Web site needs to be hosted on a Web server which is a powerful computer that can accommodate thousands of requests for the Web site pages. A Web server has to be connected to the Internet all day and night so that users can access it anytime. The high complexity and cost of maintaining a hosting platform infrastructure has resulted in a growing trend among businesses and institutions to have their applications hosted on platforms managed by another party. Such hosting platforms are referred to as *third-party hosting platforms*. A Web hosting provider is an example of a third-party hosting platform that sells space on its servers to Website owners. They provide a full-time, high-bandwidth connection to the Internet, so that visitors can access the sites easily. An example is Yahoo’s *Small Business Web hosting* service [126]. We list below some examples of the complexity and cost involved in maintaining a hosting platform:
1. Servers and software (Web server, mail server, firewall, virus protection etc.) can be expensive.

2. The server needs a 24/7 high speed connection to the Internet, which is relatively costly.

3. Setting up all the configurations including mail server, FTP server, and DNS server can be complicated.

4. Server maintenance requires twenty-four hour support, special skills, and knowledge.

Henceforth, we refer to such a third-party hosting platform as simply a hosting platform. Hosting platforms enable entrepreneurs and emerging organizations to focus on their business rather than technology. Hosting platforms are typically expected to provide performance guarantees to the hosted applications (such as guarantees on response time or throughput) in return for revenue [94]; these contracts are expressed using service-level agreements.

1.2 Models of Hosting

Due to the rapid advances in computing and networking technologies and falling hardware prices, server clusters built using commodity hardware have become an attractive alternative to the traditional large multiprocessor servers for constructing hosting platforms. Depending on the resource requirements of the applications and the strictness of the performance or resource guarantees they require, a platform may employ a shared or a dedicated model for hosting them. We elaborate on these two models of hosting applications next. Henceforth, we use the terms server and node interchangeably.

1.2.1 Dedicated Hosting

In dedicated hosting each application runs on a subset of the servers and a server is allocated to at most one application component at any given time. Dedicated hosting is
used for running large clustered applications where server sharing is infeasible due to the workload demand imposed on each individual application. In dedicated hosting either an entire cluster runs a single application (such as a Web search engine), or each individual processing element in the cluster is dedicated to a single application (as in the “managed hosting” services provided by some data centers [73]).

1.2.2 Shared Hosting

Shared hosting platforms run a large number of different third-party applications (Web servers, streaming media servers, multi-player game servers, e-commerce applications, etc.), and the number of applications typically exceeds the number of nodes in the cluster. More specifically, each application runs on a subset of the nodes and these subsets may overlap. Whereas dedicated hosting platforms are used for many niche applications that warrant their additional cost, economic reasons of space, power, cooling, and cost make shared hosting platforms an attractive choice for many application hosting environments. For example, now-a-days Web hosting is very cheap (usually starting from under $5/month). There are free Web hosting companies also that recover their costs by showing advertisements on the hosted Websites.

1.3 Internet Hosting Platform Design Challenges and Requirements

The objective of a hosting platform is to maximize the revenue generated from the hosted applications while satisfying the service-level agreements. Designing a hosting platform is made challenging by the following characteristics of Internet applications and their workloads.

Application and Platform Idiosyncrasies

1. Complex multi-tier software architecture: Modern Internet applications are complex, distributed software systems designed using multiple tiers. A multi-tier architecture
provides a flexible, modular approach for designing such applications. Each application tier provides certain functionality to its preceding tier and uses the functionality provided by its successor to carry out its part of the overall request processing. The various tiers participate in the processing of each incoming request during its lifetime in the system. Additionally, these applications may employ replication and caching at one or more tiers. These characteristics of Internet applications make inferring requirements and provisioning capacity non-trivial tasks.

2. *Dynamic content*: An increasing fraction of the content delivered by Internet applications is generated dynamically [100]. Generation of dynamic content is significantly more resource intensive than static content which accounted for the bulk of the Internet traffic a few years ago.

3. *Diverse software components*: Internet applications are built using diverse software components. For example, a typical e-commerce application consists of three tiers—a front-end Web tier that is responsible for HTTP processing, a middle tier Java enterprise server that implements core application functionality, and a backend database that stores product catalogs and user orders. These application have vastly different performance characteristics.

4. *Heterogeneous hardware*: In most hosting platforms, hardware resources get added or removed incrementally resulting in heterogeneity in the hardware.

**Internet Workload Characteristics**

1. *Multi-time-scale workload variations*: Internet applications see dynamically changing workloads that contain long-term variations such as time-of-day effects [52] as well as short-term fluctuations such as transient overloads [1]. Predicting the peak workload of an Internet application and capacity provisioning based on this estimate are known to be notoriously difficult.
2. **Extreme overloads**: There are numerous documented examples of Internet applications that faced an outage due to an unexpected overload. For instance, the normally well-provisioned Amazon.com site suffered a forty-minute down-time due to an overload during the popular holiday season in November 1999. The load seen by on-line brokerage Web sites during the unexpected 1999 stock market crash was several times greater than the normal peak load, resulting in degraded performance and possible financial losses to users.

3. **Session-based workloads**: Modern Internet workloads are often session-based, where each session comprises a sequence of requests with intervening think-times. For instance, a session at an online retailer comprises the sequence of user requests to browse the product catalog and to make a purchase. Sessions are stateful from the perspective of the application.

4. **Multiple session classes**: Internet applications typically classify incoming sessions into multiple classes. To illustrate, an online brokerage Web site may define three classes and may map financial transactions to the *Gold* class, customer requests such as balance inquiries to the *Silver* class, and casual browsing requests from non-customers to the *Bronze* class. Typically such classification helps the application to preferentially admit requests from more important classes during overloads and drop requests from less important classes.

To meet its goal of maximizing revenue given the above challenges, a hosting platform needs to carefully multiplex its resources among the hosted applications. For this, a hosting platform requires the following mechanisms.

1. **Requirement inference**: A hosting platform should be able to accurately infer the resource requirements of applications. While underestimating the resource requirements of an application can cause violations of its performance guarantees (e.g.,
degraded response times), overestimation of requirements will result in wastage platform resources and revenue. Requirement inference may be based on analytical models of applications or on empirical observations.

2. **Application placement**: Application placement refers to the problem of determining where on the cluster the various components of a newly arrived application would run. It is desirable for a hosting platform to employ a placement algorithm that allows it to maximize the revenue generated by the hosted applications.

3. **Workload prediction**: Being able to predict the workloads of the hosted applications is desirable for determining their changing resource demands. This allows the hosting platform to decide which applications to divert its resources to during a given time period.

4. **Dynamic capacity provisioning**: A hosting platform should employ mechanisms to be able to dynamically change the allocation of resources to the hosted applications to match their dynamic workloads. In a dedicated hosting platform, this would mean changing the number of servers assigned to an application; in a shared hosting platform, dynamic capacity provisioning might imply changing the CPU shares (and possibly shares of other resources) of applications on some nodes.

5. **Policing**: To protect the applications from unanticipated overloads, a hosting platform should employ request policing mechanisms. A policer allows an application to discard excessive requests so that the admitted requests continue to experience desired performance even during overloads. Further, it is desirable for a hosting platform to preferentially admit more important requests during overloads—this is in accordance with the goal of maximizing the platform’s revenue.

6. **Appropriate resource sharing OS mechanisms**: A shared hosting platform needs support from the operating systems on the constituent nodes to effectively partition re-
sources such as CPU, network bandwidth, memory etc. among the hosted application components.

Additionally, a hosting platform should be robust. We elaborate on what we mean by this below.

1. Scalability: The hosted applications should be able to operate even when the request arrival rate is much higher than the anticipated workload.

2. Failure handling: The hosting platform should employ mechanisms to handle various kinds of software and hardware failures that may occur.

1.4 The Case for a Novel Resource Management Approach: Inadequacies of Existing Work

During the past decade, several researchers have contributed to different facets of the resource management problem in hosting platforms. In this section (i) we describe the problems that have been solved (and that our thesis builds on) and (ii) we argue that there are several problems that this body of work has either not addressed at all or not solved to satisfaction.

Predictable resource allocation within a single machine is a well-researched topic. Several techniques for predictable allocation of resources within a single machine have been developed over the past decade. New ways of defining resource principals have been proposed that go beyond the traditional approach of equating resource principals with entities like processes and threads. Banga et al. provide a new operating system abstraction called a resource container which enables fine grained allocation of resources and accurate accounting of resource consumption in a single server [14]. Scheduling domains in the Nemesis operating system [68], activities in Rialto [59], and Software Performance Units [116] are other examples. Numerous approaches have been proposed for predictable
scheduling of CPU cycles and network bandwidth on a single machine among compet-
ing applications. These include proportional-share schedulers such as Borrowed Virtual
Time [37] and Start-time Fair Queuing [50], and reservation-based schedulers as in Ri-
alto [59] and Nemesis [68].

There has also been work on predictable allocation of memory, disk bandwidth and
shared services in single servers. Verghese et al. [116] address the problem of managing
resources in a shared-memory multiprocessor to provide performance guarantees to
high-level logical entities (called software performance units (SPUs)) such as a group of
processes that comprise a task. Their resource management scheme, called “performance
isolation”, has been implemented on the Silicon Graphics IRIX operating system for three
system resources: CPU, memory, and disk bandwidth. Of particular interest is their mecha-

nism for providing isolation with respect to physical memory, which works by having
dynamically adjustable limits on the number of pages that different SPUs are entitled to
based on their usage and importance. They also implement some mechanisms for man-
aging shared kernel resources such as spinlocks and semaphores. Reumann et al. [60]
propose an OS abstraction called Virtual Service (VS) to eliminate the performance in-
terference caused by shared services such as DNS, proxy cache services, time services,
distributed file systems, and shared databases. VSs provide per-service resource partition-
ing and management by dynamically deciding resource bindings for shared services in a
manner transparent to the applications. Also the resource bindings for shared services are
delayed until it is known who they work for.

In our work we build on such single-node resource management mechanisms and ex-
tend their benefits to distributed applications running on a cluster.

**Current application models are too simplistic.** Most of the existing work on mod-
eling Internet applications has looked at single-tier applications such as replicated Web
servers [36, 23, 69, 3, 74]. Since these efforts focus primarily on single-tier Web servers,
they are not directly applicable to applications employing multiple tiers, or to components
such as Java enterprise servers or database servers employed by multi-tier applications. Further, many of the above efforts assume static Web content, while multi-tier applications, by their very nature, serve dynamic Web content. Although a few recent efforts have focused on the modeling of multi-tier applications, many of these efforts either make simplifying assumptions or are based on simple extensions of single-tier models [118, 91, 61].

These models are not sophisticated enough to capture the various application idiosyncrasies we had described earlier.

**Dynamic capacity provisioning has been studied only in the context of single-tier applications.** Several papers have addressed the problem of dynamic resource allocation to competing applications running on a single server. Chandra et al. [24] propose a system architecture that combines online measurements with workload prediction and resource allocation techniques. The goal of their technique is to react to changing workloads by dynamically varying the resource shares of applications. Pradhan et al. [87] propose an observation-based approach that has the goal of designing self-managing Web servers that can adapt to changing workloads while maintaining QoS requirements of different request classes. While Chandra et al. [24] consider dynamic management of CPU, Pradhan et al. [87] manage CPU and the accept queue. Doyle et al. [36] present an approach for provisioning memory and storage resources based on simple queuing theoretic models of service behavior to predict resource requirements under changing load.

All these techniques focus on resource allocation for applications running on a single server and are inadequate for platforms hosting multi-tiered applications with components distributed across multiple nodes.

**Existing policing mechanisms do not scale.** Although a lot of research has been conducted on developing admission control algorithms for Internet applications [29, 42, 62, 70, 117, 123], the issue of the scalability of the policer itself has been unaddressed. During extreme overloads, the policer units can become bottlenecks resulting in indiscriminate, class-unaware dropping of requests causing loss in revenue.
1.5 Thesis Summary and Contributions

Having discussed the shortcomings of existing work, we describe the contributions made by our thesis.

Analytical Models for Multi-tier Applications

In this thesis, we propose analytical models of multi-tier Internet applications. Modeling single-tier applications such as vanilla Web servers (e.g., Apache) is well-studied [36, 74, 102]. In contrast, modeling multi-tier applications is less well-studied, even though this flexible architecture is widely used for constructing Internet applications and services. Extending single-tier models to multi-tier scenarios is non-trivial. Our models can handle applications with an arbitrary number of tiers and tiers with significantly different performance characteristics. Our models are designed to handle session-based workloads and can account for application idiosyncrasies such as replication at tiers, load imbalances across replicas, caching effects, and concurrency limits at each tier.

Application Placement

We formulate the application placement problem that arises in shared hosting platforms. We study the theoretical properties of this problem and develop online algorithms.

Dynamic Capacity Provisioning in Dedicated Hosting Platforms

Dynamic capacity provisioning is a useful technique for handling the multi-time-scale variations seen in Internet workloads. Dynamic provisioning of resources—allocation and deallocation of servers to replicated applications—has been studied in the context of single-tier applications, of which clustered HTTP servers are the most common example. However, it is non-trivial to extend provisioning mechanisms designed for single-tier applications to multi-tier scenarios. We design a dynamic capacity provisioning approach for multi-tier Internet applications based on a combination of predictive and reactive mech-
Managing Resources in Shared Hosting Platforms

Shared hosting environments present us with some distinct resource management challenges and opportunities. In particular, unlike dedicated environments (i) we need mechanisms to isolate collocated application components from each other and (ii) it is possible to achieve finer grain multiplexing of resources. We devise an offline profiling based technique to infer the resource needs of applications and show how a shared platform may improve its revenue by careful under-provisioning of its resources.

Overload Management

We propose overload management mechanisms that allow a hosting platform to remain operational even under extreme overloads. Our mechanisms allow an application to handle request arrival rates of several thousand requests/sec.

Table 1.1 summarizes the contributions of this thesis.

1.6 Overview of Our Hosting Platform Design

We implement all our resource management algorithms in a prototype hosting platform based on a cluster of Linux machines and evaluate them using realistic applications and

<table>
<thead>
<tr>
<th>Resource Management Issue</th>
<th>Our contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application model (dedicated)</td>
<td>Multi-tier</td>
</tr>
<tr>
<td>Application model (shared)</td>
<td>Profiling based model</td>
</tr>
<tr>
<td>Application placement (dedicated)</td>
<td>trivial</td>
</tr>
<tr>
<td>Application placement (shared)</td>
<td>Theoretical properties, Online algorithms</td>
</tr>
<tr>
<td>Dynamic provisioning (dedicated)</td>
<td>Multi-tier, predictive and reactive, Virtual Machines</td>
</tr>
<tr>
<td>Dynamic provisioning (shared)</td>
<td>Multi-tier apps</td>
</tr>
<tr>
<td>Admission Control</td>
<td>Scalable policing</td>
</tr>
</tbody>
</table>

Table 1.1. Summary of contributions.
workloads. We present the architecture of our hosting platform in Figure 1.1. We also introduce some terminology that we use throughout this thesis.

Our hosting platform consists of two main components—the control plane and the nucleus—that are responsible for managing resources in the cluster. The control plane manages resources on a cluster-wide basis—it implements the application models, and the algorithms for application placement and dynamic provisioning. The nucleus is responsible for managing resources on each individual node. It takes various measurements that are needed by the placement, provisioning, and policing algorithms. Architecturally, the nucleus is distinct from the operating system kernel on a node. Moreover, unlike a middleware, the nucleus does not sit between applications and the kernel; rather it complements the functionality of the operating system kernel. We describe the design of these components in Chapters 4 and 6.

As shown, an application may consist of multiple tiers. The figure shows a dedicated platform with each tier running on its own server. In a shared platform, we allow multiple application components to share a single server. The rest of the architecture is identical for both hosting models.

Figure 1.1. Dedicated hosting platform architecture.
Each application is guarded by a *sentry* which performs admission control to turn away excess requests during overloads. We elaborate on the design of a sentry in Chapter 7.

We borrow terminology from Roscoe and Lyles [94] and refer to that component of an application that runs on an individual node as a *capsule*. Each application has at least one capsule and more if the application is distributed. Each capsule consists of one or more resource principals (processes, threads), all of which belong to the same application. Capsules provide a useful abstraction for logically partitioning an application into sub-components and for exerting control over the distribution of these components onto different nodes. To illustrate, consider an e-commerce application consisting of a Web server, a Java application server, and a database server. If all three components need to be collocated on a single node, then the application will consist of a single capsule with all three components. On the other hand, if each component needs to be placed on a different node, then the application should be partitioned into three capsules. Depending on the number of its capsules, each application runs on a subset of the platform nodes and these subsets can overlap with one another in shared hosting.

1.7 Dissertation Road-map

The rest of this thesis is structured as follows. In Chapter 2, we present analytical models for Internet applications. In Chapter 3, we discuss the issue of application placement in hosting platforms. Chapter 4 considers the problem of dynamic capacity provisioning for Internet applications in a dedicated hosting environment. Chapters 5 and 6 presents resource management solutions unique to a shared hosting environment. Chapter 7 addresses overload management in hosting platforms. We conclude with a summary of our research contributions in Chapter 8.