APPLICATION MODEL-DRIVEN RESOURCE MANAGEMENT

BY

VANISH TALWAR

B.Tech, Institute of Technology, Banaras Hindu University, Varanasi, 1999
M.S, University of Illinois at Urbana-Champaign, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois
Abstract

IT systems in today’s enterprises are spread across organizations, are heterogeneous, and have high infrastructure and management costs. With the growing complexity of software and hardware, such costs are expected to rise exponentially. A new computing model termed as *Utility Systems* is emerging that *consolidates* IT infrastructure in centralized data centers, *shares* resources across users, allocates resources *on-demand* to user applications, and does *pay-per-use accounting* based on resource usage. Such a computing model is envisioned to reduce infrastructure costs through sharing of resources; as well as management costs by eliminating duplication of management processes, and of superfluous resources. However, a shared utility infrastructure leads to workloads competing for resources, dynamically changing workload demands, and a heterogeneous mix of applications. Existing resource management solutions for such systems either over-provision resources resulting in system inefficiency, or under-provision resources leading to unsatisfied service guarantees. As a result, with these resource management solutions, there exists a hindrance towards broader adoption of the utility model.

We propose model-driven resource management techniques and mechanisms that provide application QoS guarantees, while maintaining efficient utilization of resources. Our approach is based on creating application models that aid in prediction of performance and the behavior of applications. Creating models is challenging due to hard-to-predict workloads, complexity of application structure, scale of applications, competing customer requirements, and need for elaborate measurements for analysis.

We make the following major contributions. First, we present a methodology for applica-
tion model-driven resource management. We define a general application model consisting of a QoS model, dependency model, workload model, and performance model. The methodology consists of characterizing application workloads and dependencies, profiling application usage, developing statistical and analytical model equations, and extending them for dependencies.

Second, we apply the methodology to a remote desktop utility, in which the users’ desktop applications are hosted in a utility system, and the user accesses the remote desktop session through a thin client. We have created novel models for remote desktop sessions. The modeling derivation for a desktop session is split in two stages. In the first stage, the derivation process is applied to each application to be executed in the desktop session. We illustrate this derivation by case study of an e-mail application. In the second stage, we apply a timing dependency structure for a set of applications that would execute in the desktop session. Timing dependencies refers to the execution order in which applications are started within a desktop session.

Our third major contribution is the application of remote desktop models to resource management functions. We have developed admission control and resource assignment systems that rely on the remote desktop models to obtain the predicted resource allocation shares required to meet the SLA requirements. We have built a prototype implementation of the model-driven resource management framework, and conducted simulation studies that show the feasibility and benefits of the approach.

Validation experiments show that the prediction obtained through a model-driven approach results in combined user perceived performance guarantees and efficient utilization of resources.
To Goddess Saraswati,

Goddess of Learning and Knowledge
Acknowledgments

First and foremost, I would like to express my deepest gratitude and thanks to my advisor, Prof. Klara Nahrstedt, without whose efforts and guidance, this thesis would not have been possible. It has been my greatest good fortune to have Klara as my thesis advisor. I have always been filled with awe and respect by her tireless efforts to help her students in every possible way. This she does in spite of her extremely busy schedule. Her insights, vision, and continuous support has guided me through to the completion of this thesis. I express my special thanks to Klara for her flexibility and belief in me, agreeing for me to pursue PhD studies remotely, and permitting interactions through phone and remote presentations.

I would like to thank the respectful members of my thesis committee, Professors Roy Campbell, Indranil Gupta, Yuanyuan Zhou, and Dr. Dejan Milojicic, for their helpful discussions on my work. Their insightful comments during my PhD. Preliminary Examination greatly helped me to improve my work. I thank Prof. Mehdi Harandi for being very kind to grant me a leave of absence after my Masters, and his support throughout the PhD studies. Thanks are also due to Prof. Geneva Belford and the academic office staff especially Barb Cicone and Dana Kennedy for their advice, help, and support. I am also grateful to Anda Ohlsson for the administrative support provided all along the graduate studies.

The work presented in this thesis has been completed while I have been an employee of HP Labs. I would like to thank my management at HP Labs for the opportunity and support provided to accomplish this feat. I especially thank my present manager, Dr. Dejan Milojicic, who also kindly agreed to serve on my thesis committee. Dejan has been a great inspiration ever since I met him in October 2001. He has taught me to think critically and
practically while serving as a role model on how to work hard and efficiently. He is one of the most positive attitude persons I have met, and he continues to mold me to maintain such an attitude. I thank him for the support he has provided for my PhD studies and his flexibility as I tried to balance work and my studies. I also extend my special thanks to Dr. Raj Kumar, my former manager, for the helpful discussions shaping the early ideas for the thesis, his support during tough times at HP, and his well wishes all along. I thank my present and former department managers, Dr. Kumar Goswami and Dr. Tom Malzbender, and my lab directors, Dr. Rich Friedrich and Dr. Fred Kitson (now at Motorola), for their support, inspiration, and wishes in the course of my PhD studies. I am also grateful to John Sontag for an inspiring conversation in July 2002 that made a strong impact on me at that time to continue pursuing my PhD.

I have had the fortune to interact with a very talented and special people at HP Labs. The wonderful discussions, comments and suggestions have helped me to grow in my research and is well reflected in this thesis. Specifically, I would like to thank my colleagues and mentors at HP Labs in the teams I have been part of (in alphabetical order) - Martin Arlitt, Sujoy Basu, Yuan Chen, Keith Farkas, Sven Graupner, Subu Iyer, Vijay Machiraju, Sandro Rafaeli, Jerry Rolia, Akhil Sahai, Sharad Singhal, and Zhichen Xu. Numerous other colleagues have provided inspiration, some of whom are (in alphabetical order) – Sujata Banerjee, Lucy Cherkasova, Vinay Deodlikar, Rick McGeer, Partha Ranganathan, Sumit Roy, Puneet Sharma, Ratnesh Sharma, and Yuhong Xiong. I also thank Bikash Agarwalla for the companionship and the collaboration during his stay at HPL.

I thank my colleagues and management in the Embedded Software Operation at HP where I worked prior to my transfer into HP Labs. Special thanks to my managers Karen Choy and Satya Mylavarambhatla and team members (in alphabetical order) – Dongni Chen, Michael Diloreto, Vaidy Gopalakrishnan, Dinesh Haridas, Kyoung Kim, Abraham Liu, Collin Park, and Kim Rogers for their advice and encouragement for pursuing my PhD work. Special thanks also to Ramesh Chandra and Yuhong Xiong for spending their time to discuss
their PhD experience.

I am grateful to my friends – Prashant Mullick, Samarth Shah, and Chetan Shankar – who allowed me to stay at their apartments during my frequent trips to Champaign while pursuing this thesis. I thank Gokul Nadathur and Kiran Srinivasan for the wonderful friendship and companionship. A special thanks to Archana Samtani for her tips and advices throughout the graduate studies. I also thank all my colleagues in the MONET group for the friendship and warm welcome shown to me during my trips to Champaign.

Last but not the least, I would like to thank my family for their greatest support, caring, and patience during the long years of PhD. I thank them for their constant encouragement and inspiration without which all that I have achieved was not possible.

Above all, I thank the ALMIGHTY for HIS divine grace. I dedicate this dissertation to Goddess Saraswati, the personification of all knowledge.
Table of Contents

List of Tables ................................................................. xii
List of Figures ................................................................. xiii

Chapter 1  Introduction ...................................................... 1
  1.1  Motivation ............................................................... 1
  1.1.1  Overview of Utility Systems .................................... 1
  1.1.2  Hosted Applications and Services in Utility Systems .......... 3
  1.1.3  Resource Management in Utility Systems ..................... 5
  1.2  Thesis Statement .................................................... 7
  1.3  Contributions ....................................................... 8
  1.4  Outline of the Dissertation ...................................... 11

Chapter 2  Historical Perspective To Utility Systems ................. 12
  2.1  Evolution of Enterprise Computing ................................ 12
  2.2  Mainframe Time-Sharing Systems .................................. 14
    2.2.1  Workloads ...................................................... 15
    2.2.2  Scheduling System ........................................... 15
    2.2.3  Virtual Memory ............................................... 16
    2.2.4  Virtual Machines ............................................ 17
    2.2.5  Performance Management .................................... 18
  2.3  Real-Time Systems .................................................. 19
  2.4  User Models ........................................................ 21
  2.5  Discussion .......................................................... 24

Chapter 3  Methodology for Model-driven Resource Management .... 27
  3.1  Preliminaries ........................................................ 27
    3.1.1  Resource Management Problem ................................ 27
    3.1.2  Model-driven Approach ...................................... 28
  3.2  General Application Model ....................................... 29
  3.3  Model Derivation Steps ............................................ 32
  3.4  Application to Resource Management ................................ 35
  3.5  Thesis Roadmap ..................................................... 36
Chapter 4  Background Knowledge ........................................ 37
  4.1 Remote Desktop Utility System ..................................... 37
  4.2 Modeling Techniques .................................................. 38
    4.2.1 Histogram Analysis ............................................. 38
    4.2.2 Markov Models .................................................. 40
    4.2.3 Regression Models .............................................. 41
    4.2.4 Queuing Models ................................................ 43
Chapter 5  Remote Desktop Modeling: Part I ......................... 46
  5.1 Remote Desktop Model Definition ................................... 46
  5.2 Approach to Model Derivation ...................................... 49
  5.3 Derivation for Single Application Modeling - Overview ........... 50
  5.4 QoS Model ............................................................. 52
  5.5 Dependency Model .................................................... 53
  5.6 Workload and User Modeling ........................................ 53
  5.7 Application Profiling ................................................ 56
    5.7.1 Benchmark Suite ................................................ 56
    5.7.2 Resource Partitioning ......................................... 57
    5.7.3 Measurements - CPU Utilization ................................ 57
    5.7.4 Measurements - Screen Update Response Time .................. 60
    5.7.5 Measurements - Service Time ................................... 61
  5.8 Deriving Statistical Performance Models ............................ 62
    5.8.1 Utilization Relationships ..................................... 63
    5.8.2 Service Time Relationships .................................... 64
    5.8.3 Screen Update Response Time Relationships .................... 66
  5.9 Deriving Analytical Performance Models ............................. 67
  5.10 Extending Performance Model Relationships for Functional Dependencies  70
  5.11 Model Validation .................................................... 70
    5.11.1 Statistical Model Validation ................................ 70
    5.11.2 Analytical Model Validation .................................. 72
  5.12 Summary of Performance Model Equations ............................ 77
  5.13 Notation ...................................................................... 79
Chapter 6  Remote Desktop Modeling: Part II ....................... 80
  6.1 Derivation for Multiple Application Modeling - Overview ........... 80
  6.2 Timing Dependencies ................................................... 82
  6.3 Extending Performance Model Equations for Timing Dependencies ...... 83
  6.4 On-line Dependency Characterization ................................ 85
    6.4.1 Dependency Database ............................................. 85
    6.4.2 Dependency Tracker .............................................. 87
  6.5 Performance Model Evaluation ........................................ 87
  6.6 Notation ...................................................................... 90
## Chapter 7  Resource Management of Remote Desktop Sessions  
7.1 Execution Flow  ..................................................... 91  
7.2 Framework and Information Flow ........................................ 92  
7.3 Site Admission Control .................................................. 98  
7.4 Resource Assignment .................................................... 100  
7.5 Prototype Implementation ............................................. 105  
7.6 Simulation Analysis ..................................................... 107  
7.7 Notation ........................................................................ 116  

## Chapter 8  Related Work  .................................................. 117  
8.1 Computing and Storage Systems ......................................... 117  
8.1.1 Traditional Scientific Grids ........................................... 117  
8.1.2 Planetlab ................................................................. 118  
8.1.3 Clusters ..................................................................... 119  
8.1.4 Thin Client/Remote Desktop Computing Systems ............... 120  
8.2 Resource Management Systems ......................................... 121  
8.2.1 Classification By Application Domain .............................. 121  
8.2.2 Classification By Approach .......................................... 127  
8.2.3 Comparison ............................................................... 130  
8.3 Service Oriented Architecture ........................................... 130  

## Chapter 9  Conclusions and Future Work  .............................. 131  
9.1 Revisiting the Hypothesis .................................................. 131  
9.2 Contributions and Lessons Learned ..................................... 135  
9.2.1 Summary of Key Contributions ..................................... 135  
9.2.2 Lessons Learned ....................................................... 137  
9.2.3 Use of Results ............................................................ 138  
9.3 Future Work and Open Issues ........................................... 139  

## References ................................................................. 142  

## Vita ................................................................. 153
List of Tables

3.1 QoS Types ................................................................. 31
4.1 Classification of Markov Process ([1])................................. 40
5.1 Regression Parameters for Send Service Time (message size = 7.5 MB) .... 65
5.2 Regression Parameters for Screen Update Response Time (95% confidence bounds for the co-efficients) ................................ 67
5.3 Validation for Service Time Regression Model ......................... 71
5.4 Validation for Screen Update Response Time Regression Model ........... 71
5.5 Notations Used for the Remote Desktop Model ....................... 79
6.1 Illustration of the Dependency Matrix. Each Entry Stores the Values \( p_{ij}^{Sm}, p_{ij}^{Seq}, N_{ij} \) 86
6.2 Additional Notations Introduced in This Chapter ...................... 90
7.1 Examples of the Request Types ........................................ 110
7.2 Batch Job and Remote Desktop Session System Allocations Obtained Through Evaluation of Model Equations ........................... 111
7.3 Request Description .................................................... 112
7.4 Results for Day Time Experiments .................................... 112
7.5 Results for Night Time Experiments ................................... 114
7.6 Additional Notations Used in this Chapter ............................ 116
9.1 Comparison of 95th Percentile Screen Update Response Time With Different Approaches ............................................. 134
9.2 Comparison of 50th Percentile Value for Screen Update Response Time With Different Approaches .................................. 134
9.3 Comparison of the Service Time With Different Approaches .......... 134
List of Figures

1.1 Service Providers and Their Relationships. Multiple application providers subscribe to a utility infrastructure provider. Multiple application consumers subscribe to an application provider. Service level agreements apply to each such subscription. 2
1.2 Overprovisioning Wastes Computing Cycles Leading to System Inefficiency. The plot shown is for a single application running on a dedicated machine. 6
1.3 Best Effort Leads to QoS Violations but Better System Efficiency. The plot shown is for three identical sessions running simultaneously. 7
1.4 Illustration of the Hypothesis  8

2.1 Phases of Evolution in Enterprise Computing  13

3.1 Resource Management Functions  28
3.2 Illustration of the Model-Driven Approach  29
3.3 Model Derivation Steps  33
3.4 Illustration of Model-Driven Resource Management Functions  35

4.1 High Level Conceptual View of a Remote Desktop Utility  38
4.2 Histogram Analysis. (a) Resource usage is measured at regular intervals, (b) Usage distribution, and (c) Cumulative distribution function. 39
4.3 State Transition for a Markov Model ([2])  40
4.4 Regression Models from [2]  41
4.5 Elements of Queuing Systems  43

5.1 Remote Desktop Sessions  47
5.2 E-mail Environment  50
5.3 Session Latency Between Thin Client and Compute Server. min/avg/max/mdev= 0.317/0.828/47.853/3.982 ms  51
5.4 Session Latency Between Compute Server and E-mail Server. min/avg/max/mdev= 0.316/1.908/102.835/9.925 ms  51
5.5 Illustration of the QoS Model for E-mail Environment  52
5.6 Functional Dependency exists between the E-mail client and server  53
5.7 Illustration of the Hierarchical Nature of E-mail Interactive Workload  54
5.8 User Model (Markov Chain) for E-mail (no search operation)  55
5.9 CPU Utilization Time Plot for Send Benchmark  58
5.10 CPU Utilization Histogram for the \{Send\} Benchmark for Various Resource Shares ........................................... 59
5.11 CPU Utilization Cumulative Probability Distribution for the \{Send\} Benchmark for Various Resource Shares ........................................... 59
5.12 Screen Update Response Time Plot with Screen Update Index .................. 61
5.13 Cumulative Probability Distribution of Screen Update Response Time .......... 61
5.14 Standard Deviation for Screen Update Response Time .......................... 62
5.15 Utilization versus CPU Share Allocated for Send Benchmark .................. 64
5.16 Plot of Percentage Difference among various CPU Utilization Percentiles .... 64
5.17 Send Service Time vs. CPU Share Allocated ....................................... 65
5.18 Regression Curve for Send Service Time ........................................... 65
5.19 Screen Update Response Time vs. CPU Share Allocated ......................... 66
5.20 Regression Curve for Screen Update Response Time ............................ 67
5.21 Queuing Diagram for E-mail Application ........................................... 68
5.22 Cumulative Probability Distribution of Throughput (message size of 630 bytes) 73
5.23 Cumulative Probability Distribution of Throughput (message size of 123KB) . 73
5.24 Analytical Model Validation Using an Empty Message (of size 630 bytes) .. 74
5.25 Analytical Model Validation Using Message Size of 123 KB .................... 74
5.26 Ratio of the E-mail Server Response Time (Model/Observed) (Empty Message size of 630 bytes) .......................................................... 76
5.27 Ratio of the E-mail Server Response Time (Model/Observed) (Message size of 123 KB) .......................................................... 76
5.28 Comparison with Refined Analytical Model (Empty Message size of 630 bytes) 77
5.29 Comparison with Refined Analytical Model (Message size of 123 KB) ......... 77
6.1 Proposed Approach ........................................................................... 81
6.2 Timing Dependency Structures for Remote Desktop Sessions .................... 82
7.1 Functional Execution Flow .............................................................. 91
7.2 Resource Management Framework Principal Entities and Information Flow .... 92
7.3 Request Template ............................................................................ 93
7.4 Entities at the Resource Management Server and the Information Flow ........ 94
7.5 Illustration of Movement of Requests within Priority Queues ................. 95
7.6 Entities at the Compute Server and the Information Flow ....................... 96
7.7 Entities at the Monitoring Overlay ................................................... 97
7.8 Application Model-driven Site Admission Control .................................. 99
7.9 Considered Resource Assignment Framework ...................................... 101
7.10 End-to-end Network and Storage Bandwidth and Latency ..................... 102
7.11 Illustration of the Resource Assignment Mapping .................................. 104
7.12 The Testbed for the Prototype Implementation .................................... 105
9.1 Visual Illustration of Allocation to Choose that Satisfies Desired SLA for Screen Update Response Time and Service Time Using a Model-driven Approach 132
9.2 CPU Utilization with (a) Overprovisioning, (b) Underprovisioning (best-effort) 
(three sessions execute simultaneously), (c) CPU Shares decided using Models 
(each session gets a 40 percent CPU share allocation) .......................... 133
9.3 Cumulative Distribution Function for Screen Update Response Time. Com-
parison is made between Overprovisioning, Best-effort, and Model-driven Approaches134
9.4 Illustration of the Benefit of Model-driven Resource Management ........... 135
Chapter 1

Introduction

1.1 Motivation

Contemporary Information Technology (IT) systems in enterprises are widely spread across various IT organizations. Applications and management processes are often duplicated across these organizations. Further, these systems typically go through independent upgrade and procurement cycles which leads to heterogeneity over time. With the increasing scale and complexity of software and hardware, the infrastructure and management costs of contemporary enterprise IT is expected to grow exponentially. Researchers and IT architects are hence faced with the challenge to control the rising IT costs while meeting the growing user demands and requirements. As first steps, they have proposed the consolidation of IT systems within centralized data centers reducing the duplication of management processes. As next steps, it has been proposed to further reduce the infrastructure and management costs by increasing the degree of sharing of these consolidated computing and storage systems and to provide virtualized pools of resources to the end-user as a utility.

1.1.1 Overview of Utility Systems

In Utility Systems, computing sites, possibly geographically distributed, host the shared IT infrastructure - compute and storage servers. Users are allocated resources for their applications and data on-demand; and they are accounted for based on resource usage. Utility systems are characterized by a shared resource pool, they are subjected to dynamic
computing environments, and they host heterogeneous resources and workloads.

In practice, a utility system is enabled through a service provider model (see Figure 1.1). Two categories of service providers exist - Application Service Providers (ASPs) that own applications, and Utility Infrastructure Providers (UIPs) that own compute and storage servers. Application consumers sign subscriptions with ASPs to obtain access to applications as a service. ASPs in turn subscribe with UIPs to host the application services on the compute servers on-demand. Service level agreements (SLAs) are in place for each such subscription. Multiple ASPs subscribe to the same UIP, thus requiring the UIP to support sharing of multiple types of applications. Such a service provider model gives customers the ability to use applications without owning the software or the infrastructure needed to run the applications, and thereby reducing costs (capital, operational, maintenance, etc.) [3, 4]. Similar to utility companies, such as water and electricity, the end-users are given a pay-per-use pricing model allowing them to use the provided resource only to the extent required.

The utility infrastructure provider hosts compute servers, storage servers, and network elements (eg. network switches, routers, LANs, WAN connections). When an ASP requests
the UIP to host its’ application services, the UIP responds by allocating resources for the application service. Management Services subsequently deploy the application and provide the ASP with information about the hosted application instance (e.g., an endpoint handle). Virtualization techniques, such as virtual machines [5, 6, 7], and hardware virtualization (e.g., HP Superdome servers) are used to provide isolation among the application instances hosted in the UIP. When application consumers contact the ASP, they are directly given access to the hosted application through the end-point handle. It is the responsibility of the UIP to monitor and manage the resources such that the SLA requirements of the applications are maintained throughout the lifecycle.

Examples of emerging deployment of utility systems are: Next-Generation Data Centers; Scientific Grids such as CERN [8], ChinaGrid [9], WestGrid [10]; and Planetlab [11, 12]. For example, shared J2EE application hosting [13], remote rendering jobs [14, 15], online gaming [16, 17] Enterprise data centers host applications, such as multi-tier e-commerce [13], desktop, technical batch jobs [14, 15], and on-line gaming servers [16]. Scientific grids span global supercomputing centers, and they execute applications such as weather forecasting, protein unfolding, and rocket simulations. Planetlab nodes are hosted by global academic and industrial research institutions. Planetlab applications include file sharing and network-embedded storage, content distribution networks, routing and multicast overlays, and scalable event propagation [11, 12].

1.1.2 Hosted Applications and Services in Utility Systems

Utility systems host various classes of applications and services. Below, we summarize a few of them, classified by their structure and capabilities.

Client-Server. In client-server architecture, each computer or process on the network is either a client or a server. Servers are typically powerful computers or processes. Clients rely on servers for resources, such as files, devices, and even processing power. Examples of client-server architectures include e-commerce systems, publish-subscribe systems, remote
file services, print services.

**Peer-to-Peer.** Another type of network architecture is known as a peer-to-peer architecture because each node has equivalent responsibilities. Peer-to-peer differs from client/server architectures in which some computers are dedicated to serving the others. Peer-to-peer networks are generally simpler, but they usually do not offer the same performance under heavy loads. Examples of peer-to-peer architectures include file and content sharing, and distributed collaborative computing, such as SETI@HOME. Emerging decentralized web services have the capabilities to organize in a peer-to-peer manner.

**Streaming.** Streaming applications contain technologies for transferring data such that it can be processed as a steady and continuous stream. For streaming to work, the client side receiving the data must be able to collect the data and send it as a steady stream to the application that is processing the data and converting it to audio or video. This means that if the streaming client receives the data more quickly than required, it needs to save the excess data in a buffer. If the data doesn’t come quickly enough, however, the presentation of the data will not be smooth. With streaming, the client browser or plug-in can start displaying the data before the entire file has been transmitted. Examples of streaming applications are webcasts, radio and TV stations on the web, podcasts.

We are primarily interested in applications hosted by utility systems within enterprise data centers. Based on their structure and capability, these applications belong to one of the general classes as enumerated above. We now describe a few of the popular applications chosen by the functionality they provide.

**Enterprise Batch Applications** provide the functions of human resource, payroll, financial, asset, and cost accounting, as well as production operations and materials. Examples include SAP and PeopleSoft. Another application popular among digital animation companies are job rendering applications to render complex images and movies. Enterprise batch applications are typically compute and data intensive.

**E-commerce 3-tier Applications** are composed of multiple tiers. Typically, the tiers con-
sist of a web server, an application server, and a database server. Multiple instances of each of the tiers may execute so as to cater to the scale of requests and transactions. An example is an on-line book store. With the proliferation of the Web and Internet, a 3-tier e-commerce application is a prevalent application in enterprises. These applications are long-running, and complex, and present interesting challenges within utility systems.

Remote Desktop Sessions represent the remote execution of the users’ desktop applications. The end-user is provided with a thin client, that comprises of a network processor and display. Keyboard and mouse events are sent from the users’ thin client node to the remote compute node in the utility system, and the output of the applications is viewed by the end-user using the remote display technologies such as Citrix [18], HP RGS [19], Microsoft Terminal Servers [20], VNC [21]. The targeted customers for such a computing environment belong to the vertical markets, such as financial, health-care, and design automation. Examples of desktop applications that run in the utility are office applications (e.g., Microsoft Word, Outlook, Excel), financial stock broker and trading applications, and CAD/CAM applications.

1.1.3 Resource Management in Utility Systems

The allocation of utility resources to applications, such as those described in the previous section is done by resource management services. The resource management system performs design-time initial resource provisioning, dispatches requests to enforce the initial allocations, and responds to dynamic changes through run-time re-allocations. The design of a resource management system in a utility environment faces several challenges due to shared infrastructures, complexity of application structures, scale of infrastructure and applications, dynamic workload demands, and heterogeneous mix of applications. Contemporary resource management solutions in today’s utility systems typically do one of the following:

Over-provisioning allocates resources to applications assuming worst-case behavior. For example, a complete server (100 percent CPU shares) may be dedicated for that application
Figure 1.2: Overprovisioning Wastes Computing Cycles Leading to System Inefficiency. The plot shown is for a single application running on a dedicated machine.

(see Figure 1.2). It leads to strong application QoS guarantees. However, if the application consumes the 100 percent CPU shares for only a small fraction of execution time as is the case in Figure 1.2, it leads to poor overall system efficiency. By system efficiency is meant the overall utilization of the utility system resources.

**Under-provisioning.** The other alternative is to allocate resources by best-effort. For example, multiple applications could be admitted into a server machine, and allowed to consume shared resources by best-effort (see Figure 1.3). As a result, end-user perceived performance is unpredictable, and there are weak application QoS guarantees. On the other hand, at any instant of time, the system has a higher utilization compared to overprovisioning.

These described resource management solutions provide ad-hoc performance guarantees and system utilization. End-user perceived service response is unpredictable, and the resources are unpredictably utilized. As a result, these resource management solutions become a hindrance towards broader adoption of the utility model. In the next section, we describe our thesis statement.
Figure 1.3: Best Effort Leads to QoS Violations but Better System Efficiency. The plot shown is for three identical sessions running simultaneously.

1.2 Thesis Statement

Application model-driven resource management in shared utility systems will yield better combined application QoS guarantees and system efficiency, than over-provisioning (worst case) or under-provisioning (best-effort) of resources.

Figure 1.4 illustrates the hypothesis. Over-provisioning provides strong application QoS guarantees, but poor system efficiency. Under-provisioning provides weak QoS guarantees, but high system efficiency. A model-driven approach provides strong QoS guarantees and effective system utilization.

Models provide prediction of application behavior and performance which helps in achieving the hypothesis. However, creating application models faces several challenges summarized below.

**Hard to predict workloads.** Expected workloads in utility systems are dynamic, heterogeneous, and influenced by changing business demands and time of day characteristics.

**Complexity.** Applications consist of multiple heterogeneous components which have dependencies among them.
Multiple resource types. Each of the resource types - CPU, memory, network, storage - affect an application in significant ways. Models are required to capture the combined effect of these resource types on application performance.

Scale. Scale of customers can affect response from shared services (e.g., storage).

Elaborate measurements needed. Analysis of service time, response times, utilization, throughput, arrival rates, and user characteristics, to name a few, are required in aid of modeling.

We now summarize our major contributions in the next section.

1.3 Contributions

Our approach to prove the thesis statement is to develop technologies in support of application modeling, and then apply the models towards design of resource management modules in shared utility environments. To this end, we make the following contributions.
Methodology for Model-driven Resource Management

We define a general application model consisting of a QoS-, dependency-, workload-, and performance-model. We describe a methodology to derive such an application model. In summary, the proposed methodology is to (i) characterize workloads and dependencies, (ii) profile application usage in dedicated environment, (iii) forecast application behavior using statistical models on profiled data, (iv) complement and extend profiled relationships with analytical models, (v) extend the relationships in presence of application dependencies, and finally (vi) validate the derived models. Once the application models are developed, they are applied towards resource management functions in a utility system.

Deriving of Application Models in Remote Desktops

We demonstrate the general methodology of model-driven resource management for a remote desktop utility. We have created novel models for remote desktop sessions. These models provide prediction of resource usage for a desktop session. The modeling derivation for the remote desktop models is split into two stages. In the first stage, the derivation process is applied to each application to be executed in the desktop session. We illustrate this derivation by example of an e-mail application. The considered application QoS parameters are those of service time for e-mail send/receive operations and screen update response time. The workload is characterized to consist of a mix of send and receive e-mail operations with a message size distribution. The profiling is done using a send workload, and the relation of the service times and screen update response times to CPU share allocation is determined through statistical models. An analytical model is created using queuing theory to model the response time from an e-mail server.

In the second stage, we apply a timing dependency structure for a set of applications that execute in the desktop session. Timing dependencies refer to the execution order in which applications are started within a session. The derivation consists of two steps. First, we extend the performance model relations from the first stage, applying the dependency
structure, and obtain the performance model equations for the remote desktop. Second, we design mechanisms for dependency characterization. We propose analysis of system usage logs for statistical dependency characterization. A dependency matrix maintains the probability of sequential and simultaneous execution of applications, which is then used at run-time to infer the value of the dependency structure for a given set of applications.

**Design and Validation of Resource Management for Desktop Sessions**

We have designed resource management modules that are driven by the remote desktop models. Specifically, we have developed a *Site Admission Control* and a *Resource Assignment* system that rely on the remote desktop models to obtain the predicted resource allocation shares. The resource assignment problem is formulated as an on-line, multi-capacity, bin-packing problem. Multiple resource types correspond to the multiple capacities of bins. We define a weighted fitting metric that represents the fit between the resources desired by the model and the resources actually available in the system. We then use a Best Fit algorithm to fit as many requests as possible in the system.

We have built a *prototype implementation* to validate the feasibility of the design. The prototype represents a remote desktop utility system. Users submit requests for desktop sessions to the utility. The predicted system resource shares required to meet user SLA requirements are determined from the model equations. These predicted values are fed into admission control module, that makes admission decision for the desktop session. If the session is admitted, the resource assignment module assigns a share of resources specified by the model. We use Globus software, which we have extended to further support interactive sessions [22, 23].

We have also built a *simulation analysis tool* that implements the proposed resource management solution. Using the tool, we performed simulation studies to demonstrate benefits gained through a model-driven approach. Specifically, we emulate a real-world scenario by running experiments for a mixed workload consisting of batch and remote desktop
sessions [24]. Using a model-driven approach, we show the advantage of sharing a single pool of resources for both types of jobs (batch and desktop). This leads to cost saving of resources, while at the same guaranteeing application QoS guarantees, without much degradation in throughput and wait time.

1.4 Outline of the Dissertation

The rest of this dissertation is structured as follows. In Chapter 2 we present a historical perspective to utility systems. In Chapter 3, we provide methodology for model-driven resource management. Chapter 4 overviews background knowledge. In Chapters 5 and 6, we discuss remote desktop modeling. Chapter 7 presents the application of models to resource management. We discuss related work in Chapter 8. Finally, we conclude and present future work in Chapter 9.
Chapter 2

Historical Perspective To Utility Systems

In this chapter, we first discuss the phases of evolution in enterprise computing. We place the emerging trend towards utility enterprise IT systems within this evolution. As would be explained, the emergence of modern utility IT systems is reminiscent of mainframe systems. We describe how the various aspects of resource management for utility systems has been addressed historically in timesharing mainframe systems. We also present prior work on real-time systems and user models. We conclude by arguing the limitations of direct applicability of those works to the utility systems of today.

2.1 Evolution of Enterprise Computing

Figure 2.1 illustrates the phases of evolution in enterprise systems. Prior to 1990, mainframes were the dominant computing platforms. Mainframes are large and expensive computers that support either a timesharing or batch mode of operation. In timesharing mode, simultaneous users gain interactive access to the mainframe through terminals. Multiple users could share a machine; the operating system leverages a users’ idle time to service other users. In batch mode, users have no direct access to the computing service, it solely provides back office functions such as bulk data processing, industry/consumer statistics, ERP, and financial transaction processing. IBM System 360/370 is one of the popular mainframe systems. It has OS/360 as the operation system and Time Sharing Option (TSO) provides time sharing capabilities [25, 26].

Next in the evolution were the use of PCs, workstations, and distributed computing to
meet end-user computing as well as batch processing needs. A personal computer or PC is a microcomputer suitable for general purpose tasks such as word processing, programming, web-browsing, sending messages or digital documents, multimedia editing or game play. A workstation is a high-end general-purpose microcomputer designed to offer higher performance than normally found in a personal computer, especially with respect to graphics, processing power, and the ability to carry out several tasks at the same time. It is equivalent of a minicomputer [27].

In the next phase of the evolution, client-server architectures became popular. A server was designed as a high-capacity machine built for durability in 24x7 operations, typically hosting software server applications. As Internet and use of the Web became widespread, the demand on servers increased which led to its grouping into clusters. These clusters are classified into various functional categories, e.g., high availability clusters, load balancing clusters, high performance clusters [28]. However, with the proliferation of computing infrastructure such as PCs, workstations, servers, and clusters across the enterprise, infrastructure and management costs have begun to grow. This has led to the next phase in the evolution wherein the enterprise computing infrastructure is being consolidated into centralized data centers. This has been the trend within enterprises past few years and it is continuing on so within today’s enterprises. Figure 2.1 shows the future trend to be towards utility systems,
wherein the consolidated data centers would be *shared* and provided to end-users *on-demand* with a *pay-per-use* accounting.

Such modern day utility systems share similar philosophy with the mainframe systems. Both of these systems propose centralized computing centers, consolidation, sharing, and teleprocessing. Viewed philosophically, one could argue from the evolution shown in Figure 2.1 that modern day enterprise IT systems are returning to the computing paradigm prominent prior to 1990s. We hence pose the question: *What is different in modern day IT utility systems and why would the past work on mainframes not be directly applicable for such systems?*

In order to understand this better, we present in the next section a summary of the historical contributions made in mainframe time sharing systems. Subsequently, we also present the work in real time systems and user models. We then revisit the posed question and answer it in Section 2.5.

### 2.2 Mainframe Time-Sharing Systems

We select the historical time-sharing systems running on mainframes as an example of a utility system popular prior to the 1990s. As introduced in Section 2.1, time sharing systems allow the concurrent use of the resources of a mainframe system by a large number of users via terminal devices. With time sharing, a user can interact with the system to enter data or a program, to process a program, to retrieve information, and so on. The system rapidly switches between users, thereby each user appears to have the entire computer to himself. With time sharing [29],

- response time is of prime importance,
- job control statements must be processed dynamically, and
- there is rapid multiplexing between tasks.
IBM TSS/360 is an example time sharing operating system developed for IBM System/360 model 67 computer. Michigan Terminal System (MTS) is another operating system for System/360 [30]. Examples of the popular deployments of time sharing mainframes are PLATO at the University of Illinois [31], and ARGOS (ARGonne Operating System) [32] which runs on a Xerox Sigma 5 hardware configuration.

In the next few subsections, we describe the typical workloads and the various aspects of resource management for mainframe time sharing systems.

2.2.1 Workloads

Typical terminal users used the mainframe timesharing systems for the following types of activities:

- Software development - editing, compiling, and debugging programs. The vast majority of mainframe application programs were written in COBOL (Common Business Oriented Language). FORTRAN was the popular language in the scientific community.

- Office document production.

- File manipulation, data entry, information retrieval.

A single timesharing system served multiple users with a heterogeneous mix of such workloads. Next, we describe the scheduling for these jobs.

2.2.2 Scheduling System

We mentioned earlier in this section that response time is of prime importance in timesharing systems. Scheduling algorithms are a key to achieving a good terminal response. We describe the algorithms used by IBM TSS for achieving the same [29]. These scheduling systems permit each task to have control of the CPU for a short burst of time only, called time slice. This allows programs from multiple users to run concurrently.
In the early TSS systems, round robin type of scheduling was used. All tasks were serviced on a first-come first-served basis using a single queue. When a task reached the end of a time slice or it could no longer use the CPU, it was placed at the end of the list and the next task on the queue was given a slice of CPU time. The time slice duration was either fixed for every job or defined by function, e.g., until a certain amount of work is performed such as the completion of a compilation. As another example, when the scheduling algorithm was designed to work with a paging system, the time slice duration was chosen so as to reduce the total amount of paging that is performed by the system.

In later versions of TSS, a table driven scheduler was used. This scheduler allowed an installation to readily change scheduling parameters to meet the operational needs and dynamic characteristics of a task. A schedule table was defined that contained parameters such as time slice duration, time between time slices, number of pages allowed, priority etc. When a task entered the system, it was assigned a level corresponding to a row in the schedule table. The tasks in that level were time-sliced. When time slice end was reached, the pages were released and the task was classified as being compute-bound or paging-bound. Based on this classification, the system specifies the task’s level for its next time slice. Paging-bound tasks were typically ordered ahead of compute-bound tasks. Such a scheduling system enabled the system to respond to the dynamic requests received from terminal users.

We next explain how memory was handled.

2.2.3 Virtual Memory

The dynamic basis of user requests implied that the programs/data needed to be loaded/unloaded dynamically. Further, the loaded programs/data could be of any size. The size of main storage available on an IBM 360/67 using TSS was however only 512K to 2048K bytes [29]. In order to support the dynamism and the flexibility to support programs/data of any size, the concept of virtual memory was introduced. Virtual memory gave the user an
address space as large as the addressing capability of the computer. The memory allocation per user was managed through the use of page tables and paging. When the user referenced a page in his virtual memory, the hardware checked to see if it is main storage. If it is, then execution continued. If not, then an interruption took place and the Supervisor brought the needed page into main storage so that it could be used. Virtual memory management was one of the critical components in mainframe time sharing operating systems. For more information on the subject, the reader is referred to [33, 29].

2.2.4 Virtual Machines

Virtual machines was introduced for mainframe timesharing systems as one of the methods to increase the utilization of computers. Examples include the CP-67/CMS and later VM/370/CMS systems developed for IBM System 360 Model 67 and later System/370. With a virtual machine, each remote user appears to have a complete, dedicated computer at his disposal [34, 35]. Multiple virtual machines executed on a single mainframe. Each user could select the operating system to run in his virtual machine.

A virtual system was comprised of an operator’s console represented by the remote terminal, a virtual memory, a virtual CPU, and virtual input/output devices. The configuration of the virtual machines on a single mainframe differed from each other in terms of virtual memory size and number and type of input/output devices. The virtual machine software was composed of two independent components – control program (CP) and the operating system. Control Program (CP) provided the functions of time-sharing and resource allocation of virtual machines that allowed simultaneous operation of several virtual machines. The operating systems supported were versions of DOS/VS, OS/MFT, OS/MVT, DOS/VS. OS/VS1, SVS, MVS. One specialized system was the Cambridge Monitor System (CMS) which was designed as a single-user, conversational monitor system for terminal operations. It interpreted a simple command language typed in at the remote terminal. The commands belonged to five categories: file manipulation, compilation, execution control, debugging
aids, and utilities. CMS supported a single user at a single terminal, rather than supporting multiple terminals. For multiple terminals, one would execute multiple CMS virtual machines - one for each interactive user.

We next describe the performance management mechanisms used for mainframe time-sharing systems.

2.2.5 Performance Management

Numerous efforts were made for performance management of mainframe time-sharing systems. In this section, we summarize those efforts. The techniques described below were used to evaluate operating system algorithms, to determine OS configuration parameter values, and for capacity planning [2, 36, 37].

Analytical Models. [1, 38, 39, 40, 41, 42] Queuing theory was used to model computer systems for performance, such as to estimate response time for user requests. The queuing models developed primarily modeled the queuing effects at the processor (CPU) and storage (I/O). Queuing theory results were applied to the queuing system to obtain analytical relationships for wait times and queue lengths to CPU load, file I/O load, swap I/O load. The arrival rate of tasks per terminal user and the service time at the computing system were important input measures. Various queuing systems were developed for single server, multiple server, multi-programming, and interactive closed queuing systems. Queuing theory was also applied to model the wait time to use an on-line terminal; to estimate buffer storage requirements at message switching centers; and to estimate the effect of assigning priorities in an inquiry system.

Simulation Models. [2, 43] Analytical models were found to lack the flexibility to permit a number of different system or algorithmic modifications to be investigated with a minimum of additional effort. They also had limited applicability and were subject to limitations of mathematics. Simulation was used as an alternative. Most of the simulation models developed for mainframe time sharing systems were for system simulation wherein the basic components
- CPU, main storage, channels, control units, disk and drum storage devices, tape drives, unit record equipment were simulated. The modeler specified the various scheduling and dispatching algorithms, data management characteristics, interruption-handling disciplines, and other operating system functions. The performance of the system under different configurations of these operating system algorithms was measured and evaluated. Higher order, special-purpose simulation languages were used by the modeler to develop a simulation, e.g., SimScript by the Rand Corporation [44] and GPSS by IBM [45]. These languages have built-in facilities for time advancing, event scheduling, random number generation, statistical data collection etc.

**Experimental Evaluation.** [2, 46, 41] While analytic or simulation models assisted in performance evaluation, in most cases only actual running tests of the various alternatives provided definitive answers. Evaluation was conducted either using a controlled workload, or by running the system in an actual uncontrolled working environment. The typical performance evaluation studies conducted were for evaluating operating system algorithms, e.g., scheduling and paging. Benchmarks were created representing user tasks, thereby experiments were run varying configuration parameters such as time slice length, page replacement algorithm, scheduling priority levels, page size etc. The user behavior was measured in terms of response time and throughput. The use of statistical approach to computer performance modeling was not used very often. This is pointed out in [47] which proposes the use of a hazard function that uniquely describes the probability distribution function (p.d.f) and other distribution functions.

In the next two sections, we describe work done on real-time systems and user models.

### 2.3 Real-Time Systems

Real-time systems are required to complete the work and deliver service on a timely basis [48]. The need for such systems has historically always been there [49]. Examples include
digital control systems (including process control), command and control, data acquisition systems, signal processing, telecommunication systems, and multimedia applications. Real-time systems face the challenge of providing strict QoS guarantees to end-users. Several resource management solutions for real-time systems have been built. A good reference is [48] from where we summarize a few concepts in the next paragraph.

The jobs and processors in real time systems are characterized by temporal parameters, data dependencies, and control dependencies. Examples of temporal parameters are release times, deadlines, and timing constraints such as response time. Data and control dependencies constrain the order in which jobs can execute. The key to meeting deadlines is real-time scheduling. The commonly used approaches to real-time scheduling are the clock driven approach, weighted round-robin approach, and priority-driven approach. In a clock-driven approach, decisions on what jobs execute at what times are made at specific time instants. These instants are chosen apriori before the system begins execution. A schedule of the jobs is computed off-line and stored for use at runtime. A weighted round-robin approach builds on the basic round-robin scheme. Rather than giving all the ready jobs equal shares of the processor, different jobs are given different weights corresponding to the fraction of CPU time allocated to the job. This algorithm has been used for scheduling real-time traffic in high-speed switched networks. In a priority-driven approach, priorities are assigned to jobs. Jobs ready for execution are placed in one or more queues ordered by the priorities of the jobs. At any scheduling decision time, the jobs with the highest priorities are scheduled and executed on the available processors. A way to assign priorities to jobs is on the basis of their deadlines. In particular, the earlier the deadline, the higher the priority. The algorithm based on this priority assignment is called the Earliest-Deadline-First (EDF) algorithm.

The topic of real-time scheduling and the development of real time operating system was undertaken in the early days as well. For example, MERT is a multi-environment real time operating system for the DEC PDP-11/45 and 11/70 computers [50]. It was built on top of a kernel which provided the basic OS services. Real-time response was achieved
by means of preemptive priority scheduling. The file system structure was also optimized for real-time response. MOSS is a real-time operating system based on hierarchical levels of system functions overlayed dynamically by asynchronous cooperating processes carrying out the system activities [51]. Eleven hierarchical layers are defined each containing one or more partitions. These layers deal with timer management, processor management, channel management, memory management, event management, program management etc. Other works have addressed scheduling partially ordered tasks with probabilistic execution times [52], selecting scheduling rules that meet pre-specified response time demands [53], and building portable real-time operating systems [54].

2.4 User Models

Interactive computing systems such as time sharing systems are user driven. Modeling the behavior of the user has been historically considered important for a meaningful workload characterization. The characterization is useful for systems management of the computing center as well as for application design. In this section, we describe user behavior characterization conducted for development workloads [55] and for a computer assisted instruction application [56, 57, 58, 59, 60, 61].

The main unit of activity of a development user considered in [55] is called session – this typically starts with a log-on procedure entered by the user and terminates with a log-off procedure. Within such a session typical subtasks were performed, such as start of production runs, program and data editing. The characterization is based on a stochastic model at the task level. Each session has a hierarchical structure – each session consists of a sequence of runs (or jobs), each job consists of a sequence of tasks such as compilations, editing, executions and so on. Each task consists of a sequence of commands or statements such as insert, search, delete within an editor call. Execution of each command or statement causes a sequence of consumptions of different physical resources in the computer system,
like CPU, disk etc. The study presented in [55] collected data from a UNIVAC 1100/81 computer system. The states identified in a run were START, FOR, EDIT, MAP, FILE, SUSRES, PRSP, XQT, FIN. A Trellis diagram was used for representing the probabilistic transitions among the states.

PLATO stands for Programmed Logic for Automatic Teaching Operations. It was an interactive computer system developed at the Computer-based Education Research Laboratory (CERL), University of Illinois and made available world-wide by Control Data Corporation (CDC). It brought computer assisted instruction (CAI) to hundreds of students at once. PLATO had a large central mainframe timesharing computer with many terminals connected to it. Each terminal consisted of a keyset, a display panel, and the electronics necessary for communicating with the central computer. The computer communicated to the user through a display panel capable of displaying text and graphics. PLATO systems were used by colleges and universities, elementary schools, high schools, community colleges, medical colleges, government and business installations, and other installations. Terminals were connected to the computer by telephone lines, and so may be located anywhere. In October 1981, the UI PLATO system supported approximately 1300 terminals at over 200 locations, 44 on the campus of the University of Illinois and the others scattered throughout the United States including Hawaii [57].

There were three types of users of PLATO: Students - studied lessons assigned by the teachers. They navigated through a menu consisting of the options – AIDS, bulletin board, catalog of lessons, notes, and personal notes. AIDS was for quick reference. The personal notes were used for interactive communication. It was precursor to the electronic mail (e-mail) system. Instructors - assigned lessons to students and examined data showing the students progress. They also used the communications facilities, and studied lessons. Authors - created lesson material as well as assigned lessons to students and examined student data. They also used the communications facilities, and studied lessons. The authors used TUTOR programming language to build lessons. TUTOR was a powerful language having
more than 300 commands; yet it was relatively easy to learn, and sophisticated instructional
lessons could be written using only a small number of commands. The list of programs
created belonged to the categories of tutorial logic, inquiry logic, combination logics, drill
and practice logics.

Experimental evaluations were conducted for the usage of PLATO system. The subjects
chosen were undergraduate students at the University of Illinois. In one such study, an
experiment was designed, using CS 103, an introductory computer programming class for
students in the behavioral and social sciences [62]. The lesson series used was that of
FORTRAN. The series consists of 15 lessons (12 till the time of the study). The lessons
consisted of displays, diagrams, textual material, and quizzes. Students interacted with the
lessons to learn the material and were quizzed over it during the lessons. The purpose of the
study was to characterize the time spent by the users in the lessons, and also to compare the
performance and satisfaction level of students who used PLATO and those that did not. The
study found that the average time spent per student for required lessons was 184.5 minutes,
the number of times entered was 32.6, and the number of lessons is 9.9. The study also
characterized a breakdown of time spent for each individual lesson. These detailed results
are available from [62].

In another study, a videotape evaluation of an interactive exam system using PLATO
was conducted [63]. The experiment was conducted on non-major underclassmen from a
wide variety of fields. These students had undeveloped typing skills, minimal exposure to
interactive systems, and little motivation to learn computer programming. Each user was
limited to 10 TIPS (thousands of instructions per second), although it was found that good
response cannot be achieved if the user wants more than 3 TIPS. This was adequate for sim-
ple question-and-answer interaction but it could not support sophisticated versions of data
base search, program text analysis, or exam grading. The terminal system was found to
take 10 seconds to display a full page of text. The exam consisted of four PG/G’s (Problem
generator/graders): arithmetic expressions, FORTRAN syntax, Print with FORMAT, DO
loop. The course was CS 101, an introduction to FORTRAN for engineering students, Fall 1975. Four students were video taped and where the resolution was insufficient, the students’s activities were manually recorded. The users’ activities were categorized as follows: *Think, Answer, Problem Selection (Select)*, *Problem Generation (Generate), Load Character Set (Load), Problem Presentation (Display), What Next (subject confused) (Trouble)*. The detailed times for taped subjects was also recorded. The results for 4 subjects are as follows. The average total think time was found to be 13.24 minutes, the answer time was 1.15 minutes, the generate time was 30 second, display time was 1.17 minutes, load time was 19 seconds, the select time was 2.18 minutes, the trouble time was 2.56 minutes. Total time for the test was 21.53 minutes. The detailed productive times classified by problem type was also recorded (expressions, syntax, PRINT, DO). These detailed results are available from [63].

## 2.5 Discussion

We now revisit and answer the question posed in Section 2.1: *What is different in modern day IT utility systems and why would the past work on mainframes not be directly applicable for such systems?* We make the following arguments.

**First,** modern utility systems are built as a pool of distributed commodity servers as opposed to a single monolithic machine that mainframes use. Commodity operating systems are installed on each individual server. These commodity OS’s perform the resource management functions of scheduling, virtual memory, file systems etc. for that individual server alone. A utility operating system is provided as middleware on top of these commodity OS’s. This utility OS performs the resource management functions, e.g. resource assignment at the site level and at a coarser granularity – that of machines rather than per-CPU clocks. The design of such site level resource management functions for a distributed system faces challenges different from those considered while designing the resource management functions for a
monolithic mainframe machine.

**Second**, the workloads, structure, and design of modern applications are a lot more sophisticated and complex than those prior to 1990. An interactive workload for a typical modern application consists of several sub-tasks, and a terminal session consists of multiple applications. These sub-tasks and applications are associated with a complex state transition diagram.

**Third**, the performance management exercises conducted with mainframes were primarily targeted at the system level. The techniques used – queuing models, simulations, performance evaluations, were primarily targeted to evaluate operating system algorithms, or to model queuing effects at the processor or the I/O device. In modern day utility systems, commodity OSs would run on each individual compute/storage server. The utility system designer is not necessarily concerned with the evaluation and design of the commodity OS, but rather it relies on the commodity OS supplier, e.g., Microsoft for Windows to do so. Even if he would be interested in modeling the lower level queuing effects, e.g., processor queue, the far greater complexity of the OS functions and the application workload structure and arrivals, makes the problem non-trivial. In this thesis, we do our modeling at the application level.

**Fourth**, modern day utility system relies on commodity operating systems, and does not necessarily assume the flexibility to have specialized schedulers. This is unlike the work done in real time systems wherein a specialized scheduling algorithm is used to provide performance guarantees. Rather, in order to provide performance guarantees, the scheduling algorithm at the site level in modern-day utility system needs to rely on the ability to control the system configuration parameters. Modeling is a desirable mechanism that aids in providing estimates for these configuration parameters.

**Fifth**, today’s environments are subject to a greater dynamism. The highly distributed and interdependent nature of the utility system exacerbates the problem. Events such as plug-in and plug-out of infrastructure/applications, workload variations, and failures are common-
place in modern complex IT systems.

**Sixth, the network infrastructure and the traffic sharing patterns today are very much different than those of the mainframe systems.**

**Finally, the mainframe systems were not very efficient and cost-effective.** Todays’ IT systems are faced with mounting infrastructure and management costs. Thereby, one of the critical goals for the resource management designs within modern utility systems is to ensure good system efficiency and cost-effectiveness – better than that achieved by mainframes. This is particularly challenging given the distributed nature, complexity, and a much greater usage expected for modern IT utility systems.

With these arguments, we present in the remainder of this thesis, our contributions in the area of model-driven resource management for modern IT utility systems.
Chapter 3

Methodology for Model-driven Resource Management

In this chapter, we present an overview of the general methodology for model-driven resource management. We begin by presenting the resource management problem and the model-driven approach. We subsequently define the model, its derivation steps, and application to resource management.

3.1 Preliminaries

3.1.1 Resource Management Problem

Figure 3.1 shows conceptual view of a utility system, and its resource management functions, e.g., dynamic provisioning of resources to user requests; admission control to throttle requests; load balancing and scheduling of requests; and adaptation of allocated resources to dynamic variations in workload. Decisions made by the resource management functions affect the performance of the application, as well as the overall system utilization of the utility. Contemporary resource management systems being deployed within utility systems make decisions that lead to an over- or an under-provisioning of system resources to the applications. This results in either unsatisfied application QoS guarantees or poor system efficiency. In order to understand this better, we explain these terms next.

Application QoS guarantees are representative of SLAs agreed upon with end-customers. These guarantees can be expressed as end-user perceived parameters, such as response time, transaction time, service time, end-to-end latency, and frame rate. These would be trans-
lated to system level parameters, such as CPU utilization, network bandwidth utilization, storage bandwidth utilization.

**System efficiency** refers to utilization of system resources within the utility system. System resources are compute servers, storage servers, and network elements. Effective utilization of these resources means that the overall combined utilization of the system resources across all executing applications in the utility should be close to 100%, while at the same time meeting the application QoS requirements. Since multiple heterogeneous resources are involved, e.g., CPU, memory, and network bandwidth, the utilization measures used may have to be integrated across these different resource types.

Resource management systems need to provide combined application QoS guarantees and system efficiency for a broad adoption of the utility system concepts.

### 3.1.2 Model-driven Approach

Figure 3.2 illustrates model-driven approach to resource management to achieve the above needs. There are three key aspects of the proposed design.

First, resource controllers make use of *application models* in their decision making pro-
Figure 3.2: Illustration of the Model-Driven Approach

ceses. Models formally capture the application performance requirements enabling the decisions to be QoS-aware. Second, resource management modules are designed hierarchically, with certain resource management functionalities provided at the site level, and others distributed across individual systems. Such a design enables site level resource controllers to make decisions leveraging models of distributed application services, and with a global view of resources. Once the decisions are made, the site level resource controllers request the per-system resource managers to enforce the decisions locally. Third, the design is closed-loop. An aggregated monitoring infrastructure provides feedback on resource utilizations to the resource controllers. Decisions can now be made using the real-time state of the system. This aids for optimizing improvement in system efficiency, and enables a better use of the information obtained from models. Further, the models are determined through a process which partially involves feedback information from the resource infrastructure.

3.2 General Application Model

As introduced in the previous section, the goal of application modeling is to provide prediction of application performance. To model a given application requires its characterization
along several dimensions. While there exists general literature in the area of modeling, in practice the models derived tend to be at a very high/coarse level – usually a very simple model and/or only one modeling approach is being used. However, today’s applications and infrastructure are becoming complex with sophisticated workloads, dynamic behavior, dependencies, virtualization/partitioning, consolidation, heterogeneity, distributed nature etc. For these complex applications, one needs to model applications in a much more detail, and multiple modeling approaches would have to be used simultaneously to better characterize the application along the various dimensions. Otherwise, overprovisioning of resources will happen.

We propose a hybrid modeling approach wherein multiple modeling techniques are used. These include mechanisms to accurately model the QoS needs of applications, the structure of complex applications, the workload and user behavior, the statistical relationships, and the queuing analytical models. With such a methodology and hybrid modeling techniques, one can model larger scale and complex applications more comprehensively than before. For example, there may be some pieces of the system that to model via queuing models alone may be too expensive or infeasible, so probing and profiling and statistical inference might be better. On the other hand, there may be other pieces of the system that are shared by a large number of users and hence it may not be feasible or too expensive to derive models through experimental techniques alone, and queuing theory would help. Similarly, for user driven applications such as in a remote desktop utility, the models have to be associated with a user model. Else, the predictions made may be inaccurately applied in scenarios that vary significantly from the modeling setup. The argument for using hybrid modeling techniques is that we have now better tools to capture different dimensions of applications, and it is feasible and practical to do so.

We represent a general application model using the following tuple:

< QoS model, Dependency model, Workload model, Performance model >
**QoS model.** QoS parameters represent the performance metrics of interest. These parameters can be expressed at various levels. We consider two levels, termed *application QoS*, and *system QoS* [64] illustrated in the Table 3.1 below. We consider the application QoS model to consist of parameters, such as response time, service time, transaction time, and frame rate. The system QoS model consists of parameters, such as CPU, network, and storage utilization. These QoS parameters are expressed as *mean, peak*, and *statistical* values. The application QoS model is typically used when interfacing with users, since it represents end-user perceived performance. System QoS model is used when interfacing with system resource managers, e.g., when enforcing resource allocations.

**Table 3.1: QoS Types**

<table>
<thead>
<tr>
<th>QoS Type</th>
<th>QoS Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application QoS</td>
<td>Response time</td>
</tr>
<tr>
<td></td>
<td>Transaction Time</td>
</tr>
<tr>
<td></td>
<td>Throughput</td>
</tr>
<tr>
<td></td>
<td>End-to-end delay</td>
</tr>
<tr>
<td></td>
<td>Frame Rate</td>
</tr>
<tr>
<td>System QoS</td>
<td>CPU Utilization</td>
</tr>
<tr>
<td></td>
<td>Computation Time</td>
</tr>
<tr>
<td></td>
<td>Memory Utilization</td>
</tr>
<tr>
<td></td>
<td>Network Bandwidth Utilization</td>
</tr>
<tr>
<td></td>
<td>Storage Bandwidth Utilization</td>
</tr>
<tr>
<td></td>
<td>Network Latency</td>
</tr>
<tr>
<td></td>
<td>Storage Latency</td>
</tr>
</tbody>
</table>

**Dependency model.** Application components are related to each other through dependencies which influence the performance of the application service. A dependency model formally captures the dependency structure and parameters for an application service. The performance model for an application service relies on such a dependency model in its derivation process. Examples of dependency model categories are those of timing and functional dependencies.

**Workload model.** Performance analysis of an application service requires a characterization of workload parameters, which fall into several classes. Our primary interest is at the functional level, where the parameters are the type of application requests; the arrival rate
and ordering of these requests; and the typical characteristics of each such request, for e.g.,
the message size for a message send. Understanding these characteristics leads to a work-
load model using which one can generate a synthetic workload for performance analysis.
The workload model thus predicts the behavior of the input to the application service. It is
a pre-requisite towards developing the performance models.

**Performance model** represents the impact of system policies, workloads, mechanisms on
QoS model. It is expressed as statistical and analytical relationships between the QoS pa-
rameters and system parameters (e.g., resource allocations, number of customers), workload
parameters (e.g., arrival rate, message size), and dependency parameters.

### 3.3 Model Derivation Steps

Below, we enumerate the general steps for model derivation illustrated in Figure 3.3. We
explain the steps broadly, and postpone description of a detailed methodology for each of
these steps until Chapter 5. At that time, we explain the steps by example.

**Step 1: Identify QoS Parameters.** The QoS model signifying the application QoS and
system QoS parameters of interest is identified. As stated earlier in the chapter, applica-
tion QoS parameters are representative of SLAs. The definition and negotiation of SLAs
could involve business level decisions. Wherever such business complexities are desired to
be avoided, one can define application QoS parameters appropriate for the application, and
assume that the business SLAs would eventually translate to the considered application QoS
parameters. The considered system QoS parameters are influenced by the nature of the ap-
lication, e.g., compute-, network-, and data-intensive. From a practical standpoint, it is
also influenced by the available measurement tools and the allocation support provided by
the per-system resource managers. For example, if there is no support for enforcing mem-
ory allocations, memory has to be dropped from the list of system QoS parameters. This
decision is dependent on the modeling requirements and the specific resource management
functions it is targeted at.

**Step 2: Categorize dependency structure.** This step requires one to determine the dependencies that exist among individual application components. The main purpose of this step is to categorize and identify the dependency types. In a later step, this categorization is applied to the performance models. Part of the process to determine the dependency type could be closely related to the next step of workload characterization.

**Step 3: Identify workload and user model.** In this step, the application service in question is precisely characterized for its workload parameters. First, the application’s global workload in terms of its’ primary components is determined. Subsequently, each primary workload component is further decomposed into basic components. The basic component is then characterized by its request types, arrival, ordering, and service demand parameters. This process of workload characterization is typically undertaken through a combination of empirical techniques and domain knowledge of the application.

**Step 4: Profile application instances in a dedicated environment.** The application behavior is characterized by executing it in a dedicated environment. The workload is applied in accordance with the characterized workload model. Measurement tools are used to
gather metrics data, such as resource utilizations, service times, and response times. The workload parameters and the environment parameters, such as CPU share allocations, network bandwidth allocated etc. is varied during the profiling.

**Step 5: Forecast application behavior based on profiling.** The previous step of profiling provides measurement data, which is analyzed and characterized in this step to obtain statistical relationships. These relationships capture the impact of changes to resource allocations, workload, and environment parameters on the QoS model. Statistical techniques are used for this purpose.

**Step 6: Complement and extend profiled relationships with analytical models.** The previous step on profiling uses statistical techniques to predict application behavior. Additional relationships could be inferred using analytical modeling techniques. These modeling techniques have to consider contention for resources and the queues. The various queues in the system may be interconnected, giving rise to a network of queues. Analytical models are typically based on a set of formulas.

**Step 7: Extend the relationships for application dependencies.** The dependency structure inferred in Step 2 is used to extend and combine the performance model relations from the previous two steps. The exact methodology to use is specific to dependency type and structure. We will explain more details on this in Chapter 5.

**Step 8: Model Validation.** Once the models are created, they are validated for accuracy. For this step, the measured behavior of the application is compared with that predicted by the model.

We would like to point out a couple of points with respect to the derivation steps. *First*, Steps 3-5 are required to derive statistical models. If one does not desire statistical models, these steps may be skipped. This is indicated by the dashed arrow shown from Step 2 to Step 6 in Figure 3.3. *Second*, it is important to note that the statistical relationships derived in Step 5 are specific to the workload identified in Step 3. In case the workload of interest changes, Steps 3-5 would be repeated. This is indicated by the solid arrow from Step 5 back
to Step 3 in Figure 3.3. Such a loop may be used at runtime to refine the models in response to dynamic variations of the application workloads.

### 3.4 Application to Resource Management

Once the models are derived, a system designer can use them for various resource management functions. Figure 3.4 illustrates the picture of Figure 3.1 with model-driven approach. For example, the models can be used for provisioning functions, e.g., admission control, resource assignment, scheduling. In here, models predict the system allocations that must be provisioned so as to satisfy given application QoS requirements. Or the models could be used for run-time re-allocations (resource adaptation), where the models are used to predict the behavior of applications in response to adaptation and control configuration parameters. The models can also be applied for off-line usage such as for capacity planning. In here, the models provide statistical estimates of the utilization behavior for typical workloads. These can then be used to plan for capacity, such that the statistical guarantees for the applications would be met.
3.5 Thesis Roadmap

The general methodology for model-driven resource management is illustrated and proven in the rest of this dissertation by example. We choose remote desktop session as the application service of interest. Supporting such sessions in utility systems is an important and emerging paradigm. It is also an important complex service. Creating application models for such sessions is a novel contribution. We create remote desktop models and apply them towards the resource management functionalities of admission control and resource assignment. In here, the models are used to determine the system allocations needed to meet remote desktop session SLAs, such as screen update response time and service times. In the following chapters, we describe our solutions in more detail. We begin by presenting background knowledge on remote desktop utility systems and modeling techniques in the next chapter.
Chapter 4

Background Knowledge

In this chapter, we introduce some background material in aid of describing our solutions in subsequent chapters. We introduce the remote desktop utility system and provide background material on modeling techniques that we subsequently leverage.

4.1 Remote Desktop Utility System

A remote desktop utility consists of a pool of consolidated compute servers that host users’ desktop applications, connected to backend storage servers which host user data (see Figure 1). Users connect to the compute servers using remote display technologies, such as Citrix, Microsoft’s RDP, and VNC. The data is typically stored in central storage centers, and the individual domains maintain a local cache of the data. Remote desktop utilities extend the vision of remote display technologies by providing the compute servers as a utility and by automating the management of sessions across the pool of servers. In such a system, the user requests the utility management service for a remote desktop session. The resource management system allocates resources on-demand for the session, and it establishes the connection. Once connection is established, the end-user may interactively start one or more applications within the session. Keyboard and mouse events are sent from the users’ thin client to the remote compute server in the utility, and the output of the applications is viewed by the end-user using remote display technologies, such as VNC [21]. The paradigm of remote desktop utility system is representative of a dynamic utility computing system and has been gaining popularity in various forms. Enterprises are adopting such a system to
consolidate desktops/workstations, and thereby improve manageability, resource utilization, and save costs. The customers belong to the vertical markets, such as financial, health-care, and design automation. Example desktop applications that run in the utility are Office applications (Word, Excel, Outlook), financial stock broker and trading applications, and CAD/CAM applications.

### 4.2 Modeling Techniques

Mathematical, statistical, and state-based constructs aid in modeling performance requirements of computer systems. There are several techniques of performance modeling. In this section, we present a few of them that we leverage for our work.

#### 4.2.1 Histogram Analysis

Histogram analysis is performed on measurement data representing application behavior, such as utilization data and response time. Measurements are taken for the observed resource consumptions at regular discrete time intervals (see Figure 4.2(a)). Thereafter, the resource usage distribution is plotted as a histogram (see Figure 4.2(b)). The histogram represents the frequency of fractional resource usage within measurement intervals. The $y$ axis values
Figure 4.2: Histogram Analysis. (a) Resource usage is measured at regular intervals, (b) Usage distribution, and (c) Cumulative distribution function.

are normalized with respect to the total number of measurement intervals thus giving us probability values. A cumulative distribution function (CDF) is then plotted using these probability values (see Figure 4.2(c)). The CDF gives the probability of resource usage, i.e., \( F(x) = Pr[X \leq x] \), where \( X \) is the random variable of the applications’ fractional resource usage.

The CDF is thus representative of the cumulative probability of resource usage by the application. In order to always satisfy the applications’ requirements, we allocate the 100th percentile value of the CDF curve (i.e the resource fraction for \( F(x) = 1 \)) to the application. However, in many cases, the peak requirements are significantly higher than a high percentile of the usage [65]. Hence, one may instead provide statistical assurances by picking an appropriate percentile value from the CDF curve. For example, Figure 4.2 shows that the fraction of CPU for 100th percentile is 0.6, whereas for 95th percentile is 0.3. In such cases, we may choose a statistical QoS requirement value of 0.3. As a special case, the 50th percentile value represents the mean value.
Table 4.1: Classification of Markov Process ([1]).

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Discrete State space</th>
<th>Continuous State space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete time</td>
<td>Discrete time</td>
<td>Discrete time</td>
</tr>
<tr>
<td></td>
<td>Markov chain</td>
<td>Markov process</td>
</tr>
<tr>
<td>Continuous time</td>
<td>Continuous time</td>
<td>Continuous time</td>
</tr>
<tr>
<td></td>
<td>Markov chain</td>
<td>Markov process</td>
</tr>
</tbody>
</table>

4.2.2 Markov Models

Markov models are state-based methods for modeling. These models are useful for workload characterization and queuing analysis in support of performance modeling [2], [55], [66], [67].

A Markov process is informally defined as: given the state (value) of a Markov process $X$ at time $t$ ($X(t)$), the future behavior of $X$ can be described completely in terms of $X(t)$. Markov processes have a property that their future behavior is independent of past values.

A Markov chain is a Markov process with a discrete state space. Table 4.1 shows the classification of Markov Processes. Figure 4.3 shows a state transition diagram for a job’s transition between the CPU, disk, and terminal. As an example, a job after each visit to the CPU moves to the disk with probability 0.3, and to the terminal with probability 0.1. Similar transition diagrams can be constructed for application transitions.

![State Transition for a Markov Model ([2])](image-url)
4.2.3 Regression Models

Regression models are used to predict one variable from one or more other variables. Regression models enable the performance analyst to predict performance about past, present, or future events based on information about past or present events. The estimated variable is called the response variable, and the variables used to predict the response are called predictor variables, predictors, or factors. Regression techniques can be used to develop both linear and non-linear models (see Figure 4.4). These models are well-studied in the field of probability and statistics [2]. Examples of works that use regression models are model identification techniques in Control Theory [68, 69, 70], Network Weather Service [71], and work by Peter Dinda et al. [72]. Below, we summarize regression models.

The most common models are linear regression models. Models that limit themselves to single predictor variables are called simple linear regression models. Linear regression models the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable and the other a dependent variable. Before attempting to fit a linear model to observed data, a modeler first determines whether or not there is a relationship between the variables of interest. A valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1, indicating the strength of the association of the observed data for the two variables. A linear regression line has an equation of the form:

\[ y = a + bx, \]  

(4.1)
where \(x\) is the explanatory variable and \(y\) is the dependent variable.

The most common method for fitting a regression line is the method of \textit{least-squares}. This method calculates the best-fitting line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line (vertical deviation is zero if a point lies exactly on the fitted line). Given \(n\) observation pairs \(\{(x_1, y_1), \ldots, (x_n, y_n)\}\), the estimated response time \(y_i\) for the \(i\)th observation is:

\[
y_i = a + bx_i. \tag{4.2}
\]

The best linear model is given by the regression parameter values, which minimizes the Sum of Squared Errors (SSE):

\[
\sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a - bx_i)^2, \tag{4.3}
\]

subject to the constraint that the mean error is zero:

\[
\sum_{i=1}^{n} e_i = \sum_{i=1}^{n} (y_i - a - bx_i) = 0 \tag{4.4}
\]

The other popular technique is \textit{Autoregressive Moving Average Modeling (ARMA)} — mathematical modeling of a time series based on the assumption that each value depends only on a weighted sum of the previous values of the same series (autoregressive component) and on a weighted sum of the present and previous values of a different time series (moving average component) with the addition of a "noise" factor. If \(y(k)\) is the \(k\)-th value of the time series to model, \(u(k)\) a different time series and \(n(k)\) is noise, then ARMA model of order \((N, M)\) is given by:

\[
y(k) = \sum_{i=1}^{N} a_i y(k - i) + \sum_{i=0}^{M} b_i u(k - i) + n(k). \tag{4.5}
\]
In a typical usage of regression models, the performance usage data of an application is collected at regular intervals, then a regression model is applied to the data, and subsequently plotted.

4.2.4 Queuing Models

Queuing Models are analytical models used to describe the system performance. It models the behavior of the system as a network of queues. The modeling typically requires knowledge of system internals. Queuing models have been used in practice for several decades, and they are well described in several books [2, 1]. Some examples of queuing models are Layered Queuing Models [73], Software Performance Engineering (SPE) [74], APERA [75], work by B. Urgaonkar et al. [76], and work by Doyle et al. [77]. Below, we summarize basic material on queuing models [78].

Figure 4.5 shows the elements of a single queue queuing system: Customer Population, Arrival Process, Queue, Service, and Output. Customer Population can be considered either limited (closed systems) or unlimited (open systems). Arrival defines the way customers enter the system. The arrival is described by a random distribution of intervals also called arrival pattern. Examples of common arrival time distribution are bulk, poisson, erlang, hyperexponential, and deterministic. Queue represents a certain number of customers waiting for service. There are two important properties of a queue: maximum size and queuing
discipline. Maximum queue size is the maximum number of customers that may wait in the queue (plus the one(s) being served). Queuing discipline represents the way the queue is organized (rules of inserting and removing customers to/from the queue). Examples of queuing discipline are:

- FIFO (First In First Out), also called FCFS (First Come First Serve) — orderly queue,
- LIFO (Last In First Out), also called LCFS (Last Come First Serve) — stack,
- SIRO (Serve In Random Order), and
- Priority Queue, that may be viewed as a number of queues for various priorities.

Most models assume the normal FIFO queue. Service represents some activity that takes time and that the customers are waiting for. An important service parameter is the number of servers. Systems with only one server are called single-channel systems, systems with more servers are called multi-channel systems. Output represents the way customers leave the system. Output is mostly ignored by theoretical models, but sometimes the customers leaving the server enter the queue again (e.g., in 'round robin' time-sharing systems).

Queuing discipline is typically specified using Kendalls’ notation: \( A/B/s/q/c/p \) where:

- \( A \) is the arrival pattern (distribution of intervals between arrivals),
- \( B \) is the service pattern (distribution of service duration),
- \( s \) is the number of servers,
- \( q \) is the queuing discipline (FIFO, LIFO, ...). Omitted for FIFO or if it is not specified,
- \( c \) is the system capacity. Omitted for unlimited queues,
- \( p \) is the population size (number of possible customers). Omitted for open systems.

The symbols used for arrival and service patterns are:
• **M** is the Poisson (Markovian) process with exponential distribution of intervals or service duration respectively.

• **Em** is the Erlang distribution of intervals or service duration.

• **D** is the symbol for deterministic (known) arrivals and constant service duration.

• **G** is a general (any) distribution.

• **GI** is a general (any) distribution with independent random values.

As examples,

• **M/D/5/40/200/FCFS** represents *exponentially distributed* inter arrival times, *deterministic* service times, *five* servers, *forty* buffers (35 for waiting), total population of *200 customers*, and *first-come-first-serve* service discipline.

• **M/M/1** represents *exponentially distributed* inter arrival times, *exponentially distributed* service times, *one* server, *infinite* number of buffers, *infinite* population size, and *first-come-first-serve* service discipline.

One of the commonly used theorems in queuing theory is *Little’s Law*, which allows us to relate the mean number of jobs in any system with the mean time spent in the systems:

\[
\text{Mean number of jobs in the system } = \text{ arrival rate } \times \text{ mean response time} \quad (4.6)
\]

Little’s Law applies to any ”black box” queue under the assumptions that system is work conserving, and number of jobs entering is same as number leaving (system is stable).

Queuing models have been very popular and successful in modeling computer systems. However, they require knowledge of the internal behavior of the application. This becomes a challenge with increasing complexity and dynamically changing nature of software systems.
Chapter 5

Remote Desktop Modeling: Part I

In the next two chapters, we describe our solution towards desktop models. We begin by defining the remote desktop model using the general definition from Chapter 3. Thereafter, we explain our model derivation approach. Subsequently, we describe the derivation for single application modeling. In the next chapter, we describe the derivation for multiple application modeling.

5.1 Remote Desktop Model Definition

Figure 5.1 shows the conceptual view of a remote desktop session. For the purposes of modeling, a remote desktop session is considered as consisting of (i) remote display client hosted on a thin client machine; (ii) remote display server and one or more application processes hosted on the compute server; and (iii) storage servers serving data needed by the application processes. For example, a VNC remote desktop session consists of a VNC client, VNC remote display server, the applications running in the context of this VNC desktop session, and any data/storage services executing on a storage server that the applications access [21].

In Chapter 3, we introduced a general application model described as a tuple

< QoS model, Dependency model, Workload model, Performance model >

We now describe the definition of the instance of this tuple for a remote desktop session.
QoS model

We consider the following QoS model. The application QoS parameters are:

- service time for application workloads
- response time for remote display screen update

These parameters are representative of service level agreements with customers. The service times are specific to each application executing in the remote desktop. The values for this parameter depends on the performance of the remote display client, the remote display server, the application processes, and the data/storage processes. It must also take into account the latency of the network links. The screen update response time is the average time for the client to receive a screen update response after a remote desktop operation. The measured value for this parameter is dependent on the remote display protocol, and the system allocations made to the remote display server and application processes.

The system QoS parameters we consider are:

- CPU and memory utilization for entire desktop
- network bandwidth utilization and latency
- storage bandwidth utilization and latency
• CPU and memory utilizations per application

The performance of a remote desktop session is influenced by multiple resource types: CPU; memory; network latency between thin client and compute server; and storage access latency between compute and storage servers. The system QoS parameters are associated with the entire desktop session consisting of multiple applications, as well as with each individual application.

Dependency model

The remote desktop session is characterized by two types of dependencies that influence the performance - timing dependency and functional dependency.

Within a remote desktop session, users start applications interactively. The ordering of the start- and end-times of the applications within a remote desktop session varies dynamically, based on the interaction behavior of the user. We term such relative orderings among applications as timing dependency. The resources to be allocated for a remote desktop session very much depend on the timing dependencies.

In addition, three functional dependencies exist. First, a functional dependency exists between the remote display client and the remote display server. Requests made to the remote display client (keyboard and mouse events) are sent to the remote display server. The time for a response to those events to be displayed by the client depends upon the performance of the remote display client and the remote display server. Secondly, functional dependencies exist among application processes executing on the compute server. The processing of workloads and the ordering of application execution that are functionally dependent influences the QoS for the session. Thirdly, functional dependency exists between the application processes and the data/storage services. A workload request made to the application process that involves data would lead to a request to the data/storage services. The time for the application service to respond to the workload request is dependent on the performance of the data/storage processes.
Workload model

The global workload for the remote desktop session consists of the applications that execute in the context of the desktop session. Each of these applications is further composed of its individual application-specific operations. Each of the application operations has its own specific workload, the workload model is thus hierarchical. The specific workload parameters to be characterized at each level is application-specific, and we illustrate it by example of an e-mail application in Section 5.6.

Performance model

The performance model represents relation between the QoS parameters (service time and screen update response time) and system parameters, e.g., resource allocations made for the desktop session, number of customers, and arrival rate. We derive these relationships using statistical and analytical models.

5.2 Approach to Model Derivation

We derive the model for a desktop session in two stages. In the first stage, we illustrate the derivation process that would be applied to each application in the desktop session. In the second stage, we apply the timing dependency structure for a set of applications that execute in the desktop session. We extend the performance model relations from the first stage, and obtain the performance model equations for the remote desktop.

Accordingly, we now describe the first stage of the derivation in this chapter, and the second stage in the following chapter.
5.3 Derivation for Single Application Modeling - Overview

We show the derivation steps for single application modeling by example of an e-mail application. The setup environment consists of an end-user thin client, a compute server, and storage server. The thin client hosts the remote display client; the compute server hosts the e-mail client, remote display server, and a resource partitioning software; and the storage server hosts the e-mail server. The three nodes are connected through an interconnection network. Figure 5.2 illustrates the environment.

We first describe the considered QoS and dependency model for the e-mail environment. Subsequently, we describe the user and workload model for the setup. Then we describe profiling experiments we conducted for the characterized workload. For the profiling, we use: TightVNC client 1.2.2 as the remote display client; TightVNC server 1.2.9 as the remote display server; KDE KMail 1.4.1 as the e-mail client; Postfix Server 2.2.3 and Dovecot POP3 server 1.0.alpha as the e-mail servers. The measurements made are those for service times, screen update response time, and CPU utilization. The allocation of CPU shares is varied using a resource partitioning software, and statistical relationships among the measured QoS parameters and the CPU shares is derived using statistical techniques. We do not pursue the modeling of memory needs in depth. We have provided hooks to measure memory
consumption, but we do not study the effects of varying memory allocation on the application performance. We also ignore network latency and bandwidth for our modeling exercise. In our testbed, all of the machines are connected to a common 10 Mbps switch. Figure 5.3 and Figure 5.4 show that latency observed for a session have minimal effects, and hence we ignored it for our modeling exercise. However, we note that the same concepts we use for CPU can be applied to other resources.

We then describe the analytical modeling exercise using queuing theory. In here, the response time from the e-mail server is derived in the presence of \( N \) customers. In our modeling exercise, each of the customers is an e-mail client performing a send operation. Subsequently, we apply the functional dependency that exists among the e-mail client and the e-mail server, and combine the modeling equations obtained from the statistical and analytical exercises. Finally, we validate the statistical and analytical models, and refine them as appropriate using empirical data.

Section 5.4 – Section 5.11 now describe each of these steps in greater detail.
5.4 QoS Model

We describe in this section the specific QoS parameters identified for the e-mail environment. The application QoS parameters we consider are service time, e-mail server response time, session time, screen update response time. We explain these parameters below: (see Figure 5.5

- Service Time ($t_1$) is the time at compute server to complete a workload sequence.

- E-mail server response time ($t_2$) is the time for the e-mail client to receive response back from the e-mail server.

- Session Time ($t_3$) is the total time for a remote desktop session as perceived at end-user thin client.

- Screen Update Response Time ($t_4$) is the time measured at the end-user thin client between a keyboard (or mouse) event and the corresponding remote display screen update.

The session time, $t_3$, is inclusive of all of the activities that occur during the remote desktop session. The screen update response time, $t_4$, is an end-user perceived metric. It includes time spent at the thin client, network, compute server, and optionally storage server.
Figure 5.6: Functional Dependency exists between the E-mail client and server

The system QoS parameter we consider is that of CPU utilization at the compute server. In addition to the aggregate system usage, we also consider the utilizations of applications and the remote display server.

5.5 Dependency Model

The dependency model specific to the e-mail application is considered in terms of the functional dependency between email client and email server (see Figure 5.6). A workload request applied to the e-mail client may require the client to interact with the e-mail server. This is meant by functional dependency in this context. Hence, the total service time for the request functionally depends on the time spent at the e-mail client and at the e-mail server.

5.6 Workload and User Modeling

Modeling the workload and user characteristics is critical for the purpose of profiling the application behavior. The e-mail workload is interactive, and it can be hierarchically represented into subworkloads as shown in Figure 5.7. An e-mail workload is shown to consist of send and receive sub-workloads. Each of these sub-workloads has its own sub-sub-workloads. For example, a send workload consists of opening mail composer, subsequently typing a message, and then sending the message.

We use a Markov model to represent the transitions among the top level subworkloads, e.g., sends and receives from Figure 5.7. To be considered for performance analysis, these
subworkloads have to be the granularity of e-mail application operations. Figure 5.8 shows
the simplified Markov model for e-mail application. According to this model, the e-mail
workload can be considered as a mix of send and receive operations illustrated below.

Start S1 C R1 C R2 C S2 C S3 C R3 C S4 C R4 C R5 ... Close
where C is cycle time, user think time

This is a simplified view. In practice, other operations may also be included, e.g., search,
calendar, and configuration setting. Other important workload and user model parameters
are:

- **Probability distribution** for the number of sends (or receives) between two receives (or
  sends), e.g., the number of R’s between S1 and S2.

- **Think time distribution** between send/receive operations.

- **Message size distribution** Send and receives have a message size distribution. We
  assume that attachments are part of the message, and hence message size is inclusive
  of the size of attachments as well.
Figure 5.8: User Model (Markov Chain) for E-mail (no search operation)

- *typing speed* of e-mail messages.
- *reading speed* of e-mail messages.

Based on the user characteristics, user categories can be created, e.g., knowledge worker, high-performance worker etc., and application profiles can be classified by user type.

For the purpose of profiling, we developed our own substub-workloads for the send and receive operations, summarized below.

**Send Workload**

1. Open Mail composer
2. Fill out headers: send recipient and subject
3. Type e-mail message
4. Insert attachments
5. Send message

**Receive Workload**

1. Run Mail with check POP e-mail request
2. Go to received message
3. Open the message
4. Read message text
5. Open and read message attachments one after another
6. Close message

5.7 Application Profiling

We conduct profiling experiments on our testbed. TightVNC client 1.2.2 is the remote display client, TightVNC server 1.2.2 is the remote display server, KDE KMail 1.4.1 is the e-mail client, Postfix Server 2.2.3 and Dovecot POP3 server 1.0.alpha are the e-mail servers. All of the machines are HP Kayak XU Pentium III 500MHz, 1 GB RAM. The machines are connected through a 10 Mbps switch. A resource partitioning software, HP PRM 1.09, is used on the compute server to control the CPU allocation made for the remote desktop on the compute server. A benchmark suite that plays the send workload is applied to the KDE KMail client, and measurements are made for service times, screen update response time, and CPU utilizations. We describe the benchmark suite, the resource partitioning software, and the measurements collected in the Sections 5.7.1 - 5.7.5.

5.7.1 Benchmark Suite

The benchmarks used for the profiling is based on the send workload described in the previous section. We create micro-benchmark suites for the send operation corresponding to the workload. The benchmark suites are executed for different message sizes. For the messages, we used e-mail samples from the author’s personal e-mail box. The message size chosen for a run of the benchmark is representative of average e-mail size expected for the user class for which the profiling is performed. The benchmarks were run against KMail e-mail client application, and was recorded using VNCPlay software [79]. The measured values shown in this section are for a message size of 7.5 MB.
5.7.2 Resource Partitioning

We use HP Process Resource Manager (PRM) software to enable and enforce CPU resource partitioning [80]. A central idea behind PRM is the PRM group abstraction. Each PRM group is a conceptual partition of the system’s resources. PRM groups can be assigned a meaningful name and a share of the CPU cycles. Optionally, the administrator can also assign shares of real memory and disk I/O bandwidth. A resource share is a guaranteed share that PRM enforces to ensure that the specified policies are met. In our experiments, we assign CPU shares to Unix user IDs. Thereafter, we start a VNC server as that user id. The KDE desktop manager, and KMail client started within the remote desktop session execute as the user id. In this way, the CPU shares are enforced.

5.7.3 Measurements - CPU Utilization

We use Linux Trace Toolkit [81] as the basis for measuring CPU utilization. This toolkit records kernel system events. Examples of events recorded by the toolkit are scheduling change events, process events (e.g., fork, exit, wait), file system events (e.g., read, write, seek), memory events (e.g., allocate, free), and networking events (e.g., incoming and outgoing packets). For our CPU measurements, we collect scheduling change event traces. A snippet of a trace file for such events is show below (process traced is 29334).

Trace start time: (1128992941, 29387)
Trace end time: (1128993320, 953110)
Trace duration: (379, 923723)
...
Tracing process 29334 only

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
<th>PID</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sched change</td>
<td>1,128,992,941,290,288 29334</td>
<td>19 IN : 29334; OUT : 29437; STATE : 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sched change</td>
<td>1,128,992,941,290,450 29437</td>
<td>19 IN : 29437; OUT : 29334; STATE : 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sched change</td>
<td>1,128,992,941,290,640 29334</td>
<td>19 IN : 29334; OUT : 29437; STATE : 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sched change</td>
<td>1,128,992,941,290,751 29437</td>
<td>19 IN : 29437; OUT : 29334; STATE : 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sched change</td>
<td>1,128,992,941,290,917 29334</td>
<td>19 IN : 29334; OUT : 29437; STATE : 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
...

57
Figure 5.9: CPU Utilization Time Plot for Send Benchmark

Using these traces, we derive the CPU utilizations for the applications of interest. The methodology to do so is as follows.

**Input:** Scheduling change event traces for a process

**Output:** CPU Utilization time plots, histogram, and cumulative distribution function (CDF) for the process.

**Step 1:** Divide the trace file into intervals $I_0, I_1, ..., I_n$. Each interval is of equal length $T$.

**Step 2:** For each measurement interval $I_j$, determine the time, $T_{\text{running}}$, for which the application process is in the RUNNING state. Divide this time with the measurement interval length to obtain the utilization of the application process in that measurement interval,

$$U_j = \frac{T_{\text{running}}}{T}$$
Figure 5.10: CPU Utilization Histogram

Figure 5.11: CPU Utilization Cumulative

Probability Distribution for the \textit{Send} Benchmark for Various Resource Shares

**Step 3:** For every possible discrete utilization value, $u_i$, tally up intervals that have it as their utilization, i.e. $U_j = u_i$. Let that value be $N(u_i)$. Divide $N(u_i)$ with the total number of intervals, $n$, to give the histogram (probability of CPU usage),

$$P(u_i) = \frac{N(u_i)}{n}$$

**Step 4:** Obtain the cumulative probability,

$$F(u_i) = F(i_{i-1}) + P(u_i)$$

In our profiling, we measure the combined CPU utilization of KMail and Xvnc (the VNC server) processes. We first obtain the individual CPU utilization time plots for KMail and Xvnc (Step 2 above). We then add up the corresponding utilizations for each measurement interval. This combined utilization then represents the aggregate utilization time plot. The histogram and CDF analysis (Step 3 and Step 4) are then performed on the aggregated utilizations,
Figure 5.9 shows the time plot of CPU utilization for the send benchmark, measured for different CPU resource shares. Figure 5.10, and Figure 5.11 show the CPU utilization histogram and cumulative probability distributions (CDFs) for the send benchmark respectively, measured for different CPU resource shares. From the graph, we see that the probability of usage gets evenly spread and distributed with increase in allocation value.

5.7.4 Measurements - Screen Update Response Time

We use VNCPlay software [79] for this measurement. VNCplay is a cross-platform tool for measuring interactive performance of systems. It records a user’s interactive session with a system and replays it multiple times under different system configurations; interactive response time is evaluated by comparing the times at which similar screen updates occur in each of the replayed sessions. The main metric that we wish to obtain from the interactive replay experiments is the response time for each input event. It implements an analyzer that compares a set of replayed sessions and extracts interactive response times for various input events. The analyzer looks for similar screen updates between the replayed sessions. For example, if the user opens a menu in the recorded session, the analyzer would find the times at which the menu opened in the different replays. For each matching screen update, it finds the nearest preceding input event in all of the sessions, and assumes that this input event caused the screen update. The time difference between the screen update and the input event in each session is taken as the interactive response time for that input event in that session.

Figure 5.12 shows the screen update response time plot with screen update index. Figure 5.13 shows the CDF plot of screen update response time for different CPU shares. The response time is highest when the application is started, since it requires compute time to start the application, as well as it involves a significant amount of screen update. This can be seen by the large peaks for lower screen update indices in Figure 5.12. Figure 5.14 shows the standard deviation plots for the screen update response time. The standard deviation
Figure 5.12: Screen Update Response Time Plot with Screen Update Index

Figure 5.13: Cumulative Probability Distribution of Screen Update Response Time

reduces with increase in CPU share allocation, implying the response time smoothens with more allocation as expected. Also, the deltas in standard deviation values with increasing CPU allocation decrease, implying that there is usually a critical point in the CPU allocation beyond which the end-user’s response time experience is quite the same.

5.7.5 Measurements - Service Time

We use the Unix `date` command to measure the service times. This measurement is done on the compute server. We record the time before the start of the application operation, e.g.,

```
date +%s > sendexecutetime
```

Similarly, the time at the end of the application operation is recorded, e.g.,

```
date +%s >> sendexecutetime
```

The difference between the two times is the measured service time. This time represents the running time of the application on the compute server. In our profiling, we measure the service times for the send operations using KMail.
Figure 5.14: Standard Deviation for Screen Update Response Time

Figure 5.17 shows the service time measurements for different CPU shares for the send benchmark.

5.8 Deriving Statistical Performance Models

The profiling exercise involves executing the e-mail application in a dedicated environment (single customer), and varying the resource share allocations. The logs for the QoS metrics can now be analyzed to determine relationships for these metrics with system allocation policies. Such a relationship represents the behavior of the application. Capturing these relationships is useful for performance management so as to predict how the values of the system metrics would behave with a change in system allocation parameters. There are two approaches for representing the relationship. One is through a visual presentation that a performance analyst can manually analyze, and the other is to capture the relationship in formal mathematical relationships with the aim that it can be built into programs that can used by automated performance management tools.
5.8.1 Utilization Relationships

First, we analyze the relationship of the system QoS parameter (CPU utilization) with the CPU share allocated. Figure 5.15 plots the various percentiles for the CPU utilization of KMail+Xvnc for send benchmark. This visual representation can be used to understand the behavior of CPU utilization at various allocation points. Specifically, for example, the graph states that at 50 percent CPU share allocation, the 90th, 80th, and 50th percentile CPU utilizations for KMail+XVnc are 45 percent, 40 percent, and 8 percent respectively. This implies that 80% of KMail operations consume 40 percent or less CPU. Thus, a system administrator could use this statistical information to overbook resources, for example, by considering only the 80th percentile value, and allowing another application that consumes 10 percent or less CPU to execute simultaneously with KMail without degrading significantly the performance of KMail.

Another interesting statistical inference from Figure 5.15 is illustrated in Figure 5.16. As we increase the CPU share allocation, the percentage difference between the higher percentile value (e.g., 90th percentile) and the lower percentile value (e.g., 50th percentile) keeps increasing at an exponential rate. The percentage difference between the 90th percentile and 80th percentile keeps decreasing till 40 percent CPU shares, after which it starts increasing. If we look at the percentage difference between the 80th and 50th percentile, it increases rapidly till 40 percent CPU shares, before slowing down and then eventually decreasing after 60 percent CPU shares, thus coming closer to the 50th percentile values. Figure 5.15 thus implies that with increase in CPU share allocation, only a small fraction of requests utilize the peak available CPU utilization implying that one can obtain a good enough performance even with lower CPU share allocation. More importantly, the figure provides statistical estimates on what should the CPU share allocation be while satisfying CPU utilization needs of majority of KMail operations (between 40 and 60 percent in this case).

Finally, one also observes from Figure 5.15 that after a certain CPU share allocation, the
mean utilization (50th percentile curve) of KMail+Xvnc remains almost the same even with increase of CPU share allocated. This CPU share allocation is after 40 percent and becomes almost constant after 60 percent share, and thus it strengthens the statistical estimate in the earlier paragraph that a CPU share allocation between 40 and 60 percent should satisfy the CPU demands for the Kmail application in a remote desktop setting for the send benchmark.

5.8.2 Service Time Relationships

Figure 5.17 shows the service time measurements for different CPU shares for the send benchmark. A visual analysis of this figure shows a clear bimodal relationship. After the CPU share allocation of 60 percentile, the service time maintains a constant value. The service time values for CPU shares below 60 percentile is plotted using a regression curve shown in Figure 5.18. We performed the regression curve fit using Matlab’s Curve Fitting toolbox. This curve can be captured as a mathematical equation that can aid in its use in automated tools. A performance analyst or an automated management tool can use these
Figure 5.17: Send Service Time vs. CPU Share Allocated

results to determine the CPU shares to be allocated to satisfy service time requirements. Below is the regression equation:

\[
S_{send} = \begin{cases} 
  a \cdot e^{b \cdot \text{ResShares}} + c \cdot e^{d \cdot \text{ResShares}} & \text{if } \text{ResShares} < T \\
  S_{\text{limit}} & \text{otherwise}
\end{cases}
\]

Table 5.1: Regression Parameters for Send Service Time (message size = 7.5 MB)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a, b)</td>
<td>4611, -0.28</td>
</tr>
<tr>
<td>(c, d)</td>
<td>573.3, -0.008</td>
</tr>
<tr>
<td>(T, S_{\text{limit}})</td>
<td>60, 353</td>
</tr>
<tr>
<td>(R^2)</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 5.18: Regression Curve for Send Service Time
Figure 5.19: Screen Update Response Time vs. CPU Share Allocated

Table 5.1 shows the values for the parameters $a$, $b$, $c$, $d$, $T$, $S_{\text{limit}}$, for a fit of the curve shown in Figure 5.17.

5.8.3 Screen Update Response Time Relationships

Figure 5.12 shows the screen update response time plot with screen update index. Figure 5.13 shows the CDF plot of screen update response time for different CPU shares. Figure 5.19 shows the 95th percentile screen update response time measurements for different CPU shares. In here, we do not see a bimodal relationship. We again fit the curve using regression models. Matlab curve toolbox was used for this purpose. Figure 5.20 shows the response time regression curve for the plot of Figure 5.19. Table 5.2 shows the regression parameters.

\[ R = a \ast e^{b \ast \text{ResShares}} + c \ast e^{d \ast \text{ResShares}} \]  
(5.1)
Table 5.2: Regression Parameters for Screen Update Response Time (95% confidence bounds for the coefficients)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>393 (291.1, 1494.9)</td>
</tr>
<tr>
<td>(b)</td>
<td>-0.1274 (-0.1663, -0.0885)</td>
</tr>
<tr>
<td>(c)</td>
<td>36.11 (4.532, 67.69)</td>
</tr>
<tr>
<td>(d)</td>
<td>-0.01325 (-0.0264, -7.163 e-5)</td>
</tr>
<tr>
<td>SSE</td>
<td>13.91</td>
</tr>
<tr>
<td>(R^2)</td>
<td>0.9991</td>
</tr>
<tr>
<td>Adjusted (R^2)</td>
<td>0.9981</td>
</tr>
<tr>
<td>RMSE</td>
<td>2.153</td>
</tr>
</tbody>
</table>

5.9 Deriving Analytical Performance Models

In this section, we use queuing theory to derive analytical model for the response time from the e-mail server in the presence of \(N\) e-mail clients. Figure 5.21 shows the queuing diagram for e-mail application. The queuing system is a closed queuing system. The e-mail server is considered to consist of a send and receive server. The receive server is assumed to be a POP e-mail server, so that a user has to request the e-mail server to receive an e-mail message. Each of these servers have individual queues. The customers to the e-mail server are \(N\) e-mail clients that execute on compute servers. The workload consists of a combination of sends and receives as characterized in Section 5.6. Each e-mail operation (send or receive) waits for a response back from the e-mail server before proceeding to the next operation in the workload. This leads to a closed queuing system. We consider general service times at the e-mail server, and a general arrival rate, and hence the queuing system is a \(G/G/1\) system.

We are interested in deriving the equation for the response time from the e-mail server for a send or receive e-mail operation. Below we walk through the derivation process.
Example workload

... 
Receive
Cycle time
Some sends
Cycle time
Receive
Cycle time
Send
Cycle time
Some receives
Send
... 

Figure 5.21: Queuing Diagram for E-mail Application

Each of the send and receive servers have individual throughput measurements, \( \mu_s \) and \( \mu_r \) respectively. Since we are interested in deriving the response time for either a send or receive operation, we average the two throughputs, and use this for the average throughput of the e-mail server in our derivation,

\[
\mu = \frac{\mu_r + \mu_s}{2} \tag{5.2}
\]

Let \( S_{av} \) be the average time for an e-mail operation on the compute server. The average is taken across all sends and receive operations for all message distributions, and resource shares. Then given \( R_{server} \) as the e-mail sever response time for an e-mail operation, each e-mail client would perform an e-mail operation every \( S_{av} + R_{server} \) (ignoring user cycle time). Hence, the arrival rate for a request from a single e-mail client is \( \frac{1}{S_{av} + R_{server}} \). For \( N \) clients in the system, the arrival rate \( \lambda \) is

\[
\lambda = \frac{N}{R_{server} + S_{av}} \tag{5.3}
\]
Average e-mail server response time for an e-mail operation (send or receive) becomes

\[ R_{server} = \frac{N}{\lambda} - S_{av} \]  \hspace{1cm} (5.4)

Now, the arrival rate \( \lambda \) takes into consideration the average time for e-mail operation \( S_{av} \). However, the system arrival rate \( \lambda_{sys} \) is

\[ \lambda_{sys} = \frac{N}{R_{server}} \]  \hspace{1cm} (5.5)

Further,

\[ \rho = \frac{\lambda_{sys}}{\mu} \]  \hspace{1cm} (5.6)

Dividing equation 5.3 by \( R_{server} \), and substituting for \( \lambda_{sys} \) gives

\[ \lambda = \frac{\lambda_{sys}}{1 + \frac{S_{av}}{R_{server}}} \]  \hspace{1cm} (5.7)

\[ = \frac{\rho \mu}{1 + \frac{S_{av}}{R_{server}}} \]  \hspace{1cm} (5.8)

Solving the above gives

\[ \rho \mu R_{server}^2 + R_{server} (S_{av} - N) - NS_{av} = 0 \]  \hspace{1cm} (5.9)

Solving the quadratic equation gives

\[ R_{server} = \frac{N - S_{av} \pm \sqrt{(S_{av} - N)^2 + 4\rho \mu NS_{av}}}{2\rho \mu} \]  \hspace{1cm} (5.10)
5.10 Extending Performance Model Relationships for Functional Dependencies

We consider the functional dependency among the e-mail client and the e-mail server. Accordingly, the total service time for a send request would be the sum of (i) the time taken at the compute server, and (ii) the e-mail server response time. We derived the former using the statistical performance model equation described in Section 5.8.2, and the latter using the analytical performance model equation described in Section 5.9. Hence, the extended performance model equations for the service times are:

\[
S_{\text{send}}^{\text{extended}} = S_{\text{send}} + R_{\text{server}} \tag{5.11}
\]

\[
S_{\text{receive}}^{\text{extended}} = S_{\text{receive}} + R_{\text{server}} \tag{5.12}
\]

5.11 Model Validation

We conduct validation for the statistical and analytical model individually.

5.11.1 Statistical Model Validation

We first explain below the experimental setup, and subsequently we present and discuss the validation results.

Experimental Setup

The experimental setup is similar to that described in Sections 5.3 and 5.7 - a thin client hosts the remote display client; a compute server hosts the e-mail client, remote display server, and a resource partitioning software; and the storage server hosts the e-mail server. The software packages used are TightVNC version 1.2.2 as the remote display client and server, KDE KMail version 1.4.1 as the e-mail client, Postfix Server version 2.2.3 and Dovecot
POP3 server version 1.0.alpha as the e-mail servers. All of the machines are HP Kayak XU Pentium III 500MHz, with 1 GB RAM, and connected through a 10 Mbps switch. The resource partitioning software used is HP PRM version 1.09 that controls the CPU allocation made for the remote desktop on the compute server. We apply the send workload described in Section 5.6 to the KDE KMail client.

Validation

We validate the regression model equations described in Section 5.8 by allocating CPU shares different from those used to derive the regression equations, and comparing the observed versus the predicted service times and screen update response time for the desktop session. Specifically, we allocate CPU shares of values 70, 50, and 30 for the remote desktop session hosted on the compute server. Tables 5.3 and 5.4 show the comparison for the service time and screen update response time respectively. As can be observed, the expected and observed values are quite close. The difference increases as we reduce the CPU shares from 70 to 50 to 30 percent. We however do not have enough data to make a general statement about such a trend.

<table>
<thead>
<tr>
<th>Resource Share Allocated</th>
<th>Service Time (Model)</th>
<th>Service Time (Observed)</th>
<th>Difference (+/-)</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>353 s</td>
<td>354 s</td>
<td>1 s</td>
</tr>
<tr>
<td>50</td>
<td>384.3 s</td>
<td>364 s</td>
<td>20.3 s</td>
</tr>
<tr>
<td>30</td>
<td>452 s</td>
<td>418 s</td>
<td>34 s</td>
</tr>
</tbody>
</table>

Table 5.4: Validation for Screen Update Response Time Regression Model

<table>
<thead>
<tr>
<th>Resource Share Allocated</th>
<th>Screen Update Response Time (Model)</th>
<th>Screen Update Response Time (Observed)</th>
<th>Difference (+/-)</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>14.34 s</td>
<td>14.29 s</td>
<td>0.05 s</td>
</tr>
<tr>
<td>50</td>
<td>19.29 s</td>
<td>15.16 s</td>
<td>4.13 s</td>
</tr>
<tr>
<td>30</td>
<td>32.87 s</td>
<td>25.32 s</td>
<td>7.55 s</td>
</tr>
</tbody>
</table>
5.11.2 Analytical Model Validation

We first explain below the experimental setup, and subsequently we present and discuss the validation results.

Experimental Setup

We use Postfix server version 2.2.3 and Dovecot POP3 server version 1.0.alpha as the e-mail send and receive servers respectively. KDE KMail version 1.4.1 is the e-mail client. We execute \(n\) simultaneous e-mail sessions explained as follows. We create \(n\) scripts, each representing an e-mail session. Each session consists of alternate sends and receives with a 5 second cycle time between them. The message content is created off-line and saved in a file. The send and receive operations in the scripts use KMail’s command line options. Before executing the scripts, we start an instance of KMail. The send and the receive operation in the scripts subsequently use KDE’s \textit{dcop} feature to communicate with the running instance of KMail.

```
body='cat /home/expts/queuingscripts/file'
dcopref='dcop kmail KMailIface openComposer "mailexptreceive@quark0.test.com" \ "" "" "Subject" "$body" true'
dcop "$dcopref" send 1
```

The receive operation uses the checkMail feature of KMail to receive mail from the POP server. The command used is

```
dcop kmail default checkMail
```

These sends and receives from the email sessions are sent through a single KMail client. The reason for this is the following. It is practically difficult to execute a large number of simultaneous unique KMail clients. On a Unix system, one can only have one distinct KMail client executing per \textit{userid}. Since running an instance of KMail requires X server, it implies that to simulate \(n\) simultaneous e-mail sessions, one would have to start \(n\) X sessions if only
a single computer is available. If multiple computers are available, the number of X sessions started per computer would be less, however the mechanism tends to get inconvenient and complicated when multiple machines are used when n gets large. Instead, we execute the n e-mail sessions through a single e-mail client. We believe that the queuing effect (wait time) perceived at the end client script using such an approach is equivalent to that when n distinct e-mail clients are used. Setting up a more sophisticated experimental setup with n distinct e-mail clients on multiple machines sending and receiving messages of different sizes is planned for future work.

Throughput is measured at the e-mail server through analysis of the Postfix and Dovecot server logs. We measure the response time for each e-mail send and receive operation in an e-mail session. During our experiments, we varied the number of simultaneous client sessions from 1 upto 100. The clients are allowed to execute for 30 minutes for each value of n. During this time, several sends and receives take place, and we measure response time for each such operation. We average the response time across all sends and receives for all e-mail sessions. This average is considered as the experimentally experimental response time. Figures 5.22 and 5.23 show the throughput cumulative probability plot for few of the
client sessions.

Validation

Figures 5.24 and 5.25 show the comparison of the response times predicted by the model and that observed experimentally. In Figure 5.24, an empty message (630 bytes) is sent, whereas in Figure 5.25, relatively larger messages (of size 123 KB) are sent. The traffic intensity is assumed to be 1 in these experiments, i.e., the arrival rate $\lambda_{sys}$ is equal to the throughput $\mu$.

From the figures, one can observe that the response time predicted by the model, and that experimentally observed follow similar pattern. The difference between the experimentally observed and the model predicted values is attributed to two factors. First, the throughput value used in the evaluation of the model equation is calculated as the aggregate throughput of the entire send (or receive) execution at the e-mail server. Typically, the send (or receive) execution involves multiple processes and functions. We are interested in the throughput of the process that interfaces with the e-mail client. This throughput is expected to be more than the values we measured and substituted into the model equations. Hence, the
response time values evaluated using the model in Figures 5.24 and 5.25 are larger than
the values they should have in reality. Second, the experimentally observed values for the
response time is dependent on the internal architecture and implementation of KMail and
Postfix/Dovecat servers. These applications use sophisticated queuing mechanisms, threaded
architectures, and other optimizations. We have not undertaken a deeper investigation
into the architecture and implementation of these applications, and hence do not have a
comprehensive explanation of the values shown in Figures 5.24 and 5.25. This is planned for
future work.

Model Refinement

In the previous subsection, we provided explanation on the difference seen between the
experimentally observed and the model predicted values in Figures 5.24 and 5.25. The
explanation provided point towards a broader issue in the creation and use of models – to
factor in measurement errors and application optimizations.

In order to determine what this factor is for our models, we plotted the ratio of the model
predicted versus the experimentally observed e-mail server response times. These plots are
shown in Figures 5.26 and 5.27. We chose to plot the ratio, since a visual observation of
the graphs in Figures 5.24 and 5.25 gave us the intuition that the model predicted values
are proportionally scaled with respect to the experimentally observed values. However, in
other scenarios, one may have chosen to plot other parameters as well, such as the numerical
difference between the values. From Figures 5.26 and 5.27, one observes that the ratio is
large when the number of simultaneous client sessions are small. But, as the number of
simultaneous client sessions increase, the ratio becomes smaller and in relatively narrow
range. This is more prominent for the plot in Figure 5.27 than the plot in Figure 5.26.
We decided the adjust the model equation 5.10 by dividing it by an adjustment factor, e,
which is equal to the average ratio of the original model predicted versus the experimentally
observed values for the e-mail server response time. This is represented as

$$R_{adj}^{server} = \frac{R_{server}}{e} \quad (5.13)$$

As pointed in the previous paragraph, the ratio values shown in Figures 5.26 and 5.27 are larger when number of client sessions are small, and reduce to a more narrow range with increase in sessions. We plot the e-mail server response time with the adjusted model equation for two different values of $e$ - one in which we ignore the values when the number of client sessions is small, and the other in which we consider all the values. These are shown in Figures 5.28 and 5.29 for the two sets of validation experiments. Specifically, for the plot with message of size 630 bytes, we ignore the values when client sessions are 10, 20, and 30. The average ratio for the remaining sessions is 3.75. If we do not ignore these client sessions, the average ratio is 4.25. For the plot with message of size 123KB, we ignore the values when client sessions are 5, and 10. The average ratio for the remaining sessions is 2.75. If we do not ignore these client sessions, the average ratio is 4.25.
Figure 5.28: Comparison with Refined Analytical Model (Empty Message size of 630 bytes)  
Figure 5.29: Comparison with Refined Analytical Model (Message size of 123 KB)  

As can be observed from Figures 5.28 and 5.29, the adjusted model matches the observed values quite well. In practice, with enough empirical data over several runs of the application, one could get an approximate estimate of $e$ for the application. The model designer could then provide this estimate along with the adjusted model equations.

### 5.12 Summary of Performance Model Equations

Configuration parameters:

- message size distribution
- number of customers
- throughput of e-mail server
- resource allocation shares
- user type.
Equations are

\[
S_{\text{send}} = \begin{cases} 
    a \cdot e^{b \cdot \text{ResShares}} + c \cdot e^{d \cdot \text{ResShares}} & \text{if } \text{ResShares} < T \\
    S_{\text{limit}} & \text{otherwise}
\end{cases}
\]

\[
\rho \mu R_{\text{server}}^2 + R_{\text{server}}(S_{av} - N) - NS_{av} = 0 \quad (5.14)
\]

\[
S_{\text{send}}^{\text{extended}} = S_{\text{send}} + R_{\text{server}} \quad (5.15)
\]

\[
S_{\text{receive}}^{\text{extended}} = S_{\text{receive}} + R_{\text{server}} \quad (5.16)
\]

\[
R = a \cdot e^{b \cdot \text{ResShares}} + c \cdot e^{d \cdot \text{ResShares}} \quad (5.17)
\]

\[
R_{\text{adjusted}} = \frac{R_{\text{server}}}{e} \quad (5.18)
\]

Solving the equation for a given target send service time, receive service time, and screen update response time gives us the CPU shares that must be respectively allocated. Let us label them as \(\text{ResShares}_{\text{send}}, \text{ResShares}_{\text{receive}},\) and \(\text{ResShares}_{\text{restime}}\) respectively. Then, the final CPU resource shares that must be allocated to satisfy all three targets is the maximum of the three =

\[
\max(\text{ResShares}_{\text{send}}, \text{ResShares}_{\text{receive}}, \text{ResShares}_{\text{restime}}) \quad (5.19)
\]

Please note that the CPU share values obtained are dependent on the machine processor speed on which the profiling exercise took place. Hence, the CPU shares value is first converted to cycles/second, before using it for resource management functions. Specifically, the CPU share is multiplied by the processor speed of the machine on which the profiling was performed.
## 5.13 Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{send}$</td>
<td>E-mail client service time at the compute server to process send operation</td>
</tr>
<tr>
<td>$S_{receive}$</td>
<td>E-mail client service time at the compute server to process send operation</td>
</tr>
<tr>
<td>ResShares</td>
<td>The fractional CPU resource shares allocated at the compute server</td>
</tr>
<tr>
<td>$R$</td>
<td>Screen update response time</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Throughput of e-mail server</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of e-mail clients (customers of e-mail server) in the system</td>
</tr>
<tr>
<td>$R_{server}$</td>
<td>Response time from e-mail server</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Arrival rate of requests to the e-mail server</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Traffic intensity</td>
</tr>
<tr>
<td>$S_{extended\ send}$</td>
<td>Total service time for e-mail send operation considering the time at compute server, and response time from e-mail server</td>
</tr>
<tr>
<td>$S_{extended\ receive}$</td>
<td>Total service time for e-mail receive operation considering the time at compute server, and response time from e-mail server</td>
</tr>
<tr>
<td>$e$</td>
<td>Adjustment factor for analytical model</td>
</tr>
<tr>
<td>$R_{adjusted\ server}$</td>
<td>Response time from e-mail server refined to take into considerations adjustments from empirical observation</td>
</tr>
</tbody>
</table>
Chapter 6

Remote Desktop Modeling: Part II

This chapter is a continuation of the previous chapter on remote desktop models. In this chapter, we present our contributions towards modeling of desktop sessions containing multiple applications. The model equations are extensions of those derived in the previous chapter.

6.1 Derivation for Multiple Application Modeling - Overview

We present the model derivation steps when a set of applications are executing in a desktop session. We consider there exists a timing dependency among these applications, i.e. these applications are started and stopped in relative ordering. We ignore any functional dependencies among the application in our derivation. While functional dependencies would be common-place in desktop sessions, we only address timing dependencies in this dissertation, and consider addressing functional dependencies among applications in a desktop session as part of our future work.

The derivation process consists of two steps. In the first step, the performance model equations are formulated. These equations are considered to be extensions of the equations derived for each application within the desktop session. Specifically, we obtain equations for the resource shares needed for a desktop session, given resource share requirements for each individual application. The resource share requirement for each individual application
is obtained from the performance model equations derived in the previous chapter. The equations for the remote desktop session are a function of the dependency structure among the individual applications. This dependency structure is characterized as \textit{Sim} for simultaneous execution of applications, \textit{Seq} for sequential execution of applications, and \textit{Mixed} for a combination of simultaneous and sequential execution of applications.

The second step of the derivation process is to design mechanisms for deriving the dependency structures for a given set of applications. Our proposed approach is to use on-line characterization based on analysis of historical session process logs. Figure 6.1 illustrates the approach. A dependency tracker analyzes the dependencies from historical logs, and updates a probability matrix, termed as dependency database. This matrix stores the probabilities that a given pair of applications would execute simultaneously or sequentially.

We present these two derivation steps in detail in this chapter. Subsequently, we present an algorithm to evaluate the model equations at run-time, using the dependency information from the dependency database.
Modeling the resource requirement for a remote desktop session at the Resource Management Server. The top, middle, and bottom graphs show the Simultaneous, Sequential and Mixed execution order of five applications.

Figure 6.2: Timing Dependency Structures for Remote Desktop Sessions

6.2 Timing Dependencies

A remote desktop session consists of a set of individual applications. There exists a dependency among these applications. These dependencies exist in terms of execution orders among the applications. We term such dependencies as timing dependencies. With timing dependencies, we assume that the applications are functionally independent. Figure 6.2 illustrates our considered timing dependency structures:

**Simultaneous execution ordering** in which an instance of each application executes simultaneously. This is denoted as $Sim$.

**Sequential execution ordering** in which instances of the applications execute sequentially one after the other during the session. This is denoted as $Seq$.

**Mixed Case** when some applications are executed simultaneously, and some others are executed sequentially. This is denoted as $Mix$. 
6.3 Extending Performance Model Equations for Timing Dependencies

The applications within a remote desktop session are all started interactively by the end-user and they execute in the context of the remote desktop session on the compute server. We denote the resource requirement for the remote desktop session in the following manner

\[ Resource\ Shares_{desktop} = \{C_{desktop}, N_{desktop}, S_{desktop}, L_{N_{desktop}}, L_{S_{desktop}} \} \]  \hspace{1cm} (6.1)

For the sake of convenience, we assume in the discussion that follows that each of \( C_{desktop}, N_{desktop}, \) and \( S_{desktop} \) is representative of any of the mean, peak, or statistical values. The discussion that follows will hold true for each of these types of values. The performance model for the remote desktop session is:

\[ Resource\ Shares_{desktop} = F(D, R) \]  \hspace{1cm} (6.2)

where \( D \) is the timing dependency structure, and \( R \) is resource shares requirements vector for the individual applications, \( A_i, i = 1 \) to \( n. \)

\[ R = \{\{C, N, S, L_N, L_S\}_{A_i}, 1 \leq i \leq n\}. \]

These values are obtained using the equations from last chapter.

One can plug in various policies that determine the value of the function \( F(D,R). \) We present one such policy. Below, we enumerate the values of \( C_{desktop}, N_{desktop}, S_{desktop}, L_{N_{desktop}}, \) and \( L_{S_{desktop}} \) according to this policy for various values of \( D. \)

- \( D = Sim. \) In this case, the aggregate resource requirements for the remote desktop
session is modeled as the sum of the individual requirements

\[
C_{desktop} = C_O + \sum_{i=1}^{i=n} C_i
\]

\[
N_{desktop} = N_O + \sum_{i=1}^{i=n} N_i
\]

\[
S_{desktop} = S_O + \sum_{i=1}^{i=n} S_i
\]

(6.3)

where \(C_O, N_O, S_O\) are the extra overheads that is accounted for due to other processes e.g., monitoring software etc., that may run within the remote desktop session at runtime. The latency requirements for the remote desktop session is taken as the minimum of those for the individual application sessions.

\[
L_{N_{desktop}} = \min_{i=1}^{i=n} L_{N_i}
\]

\[
L_{S_{desktop}} = \min_{i=1}^{i=n} L_{S_i}
\]

(6.4)

• **D= Seq.** In this case, the aggregate resource requirements for the remote desktop session is modeled as the maximum of the individual requirements

\[
C_{desktop} = C_O + \max_{i=1}^{i=n} C_i
\]

\[
N_{desktop} = N_O + \max_{i=1}^{i=n} N_i
\]

\[
S_{desktop} = S_O + \max_{i=1}^{i=n} S_i
\]

(6.5)

The latency requirements for the remote desktop session is taken as the minimum of
those for the individual application sessions.

\[
L_{N_{\text{desktop}}} = \min_{i=1}^{i=n} L_{N_i} \\
L_{S_{\text{desktop}}} = \min_{i=1}^{i=n} L_{S_i} \quad (6.6)
\]

- **D= Mix.** In this case, the resource requirement is modeled as a value between the two extremes of simultaneous execution and sequential execution. For example, we evaluate Equations 6.3 and 6.5, and take the mean of the two values. More generally, we pick a percentile value between the two extremes based on the prediction strength of the dependency structure *Mix*.

- **D= Unknown.** In this case, the resource requirements for the remote desktop session would be modeled assuming worst case requirements (such as requiring the maximum permissible resources on a node), or an administrator would specify the requirements.

### 6.4 On-line Dependency Characterization

In this section, we present the dependency database and dependency tracker used for on-line dependency characterization. Figure 6.1 shows the considered approach. A dependency database stores known dependency information among applications. This database is populated with historical information that is collected by the dependency tracker. The tracker collects information from a running system and updates the database.

#### 6.4.1 Dependency Database

The information stored within the dependency database can be viewed as a matrix shown in Figure 6.1. Each entry of the matrix stores the historically observed probability value of the dependency structures *Sim* and *Seq* among pairs of applications. The matrix is represented
Table 6.1: Illustration of the Dependency Matrix. Each Entry Stores the Values 

\( P_{ij}^{Sim}, P_{ij}^{Seq}, N_{ij} \)

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3</td>
<td></td>
<td></td>
<td>(0.4, 0.6, 11)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

more formally as:

\[ M = (P_{ij}^{Sim}, P_{ij}^{Seq}, N_{ij}) \]  \hspace{1cm} (6.7)

such that

\( P_{ij}^{Sim} = Pr(\text{Applications } i \text{ and } j \text{ execute simultaneously}) \)

\( P_{ij}^{Seq} = Pr(\text{Applications } i \text{ and } j \text{ execute sequentially}) \)

\( N_{ij} \) is the total number of observations made that involved applications \( i \) and \( j \)

For example, referring to Figure 6.1, the values of \( P_{ij}^{Sim}, P_{ij}^{Seq} \), and total number of observations among applications \( x3 \) and \( x2 \) is shown to be 0.4, 0.6, and 11 respectively. This implies that out of the 11 observations in which \( x3 \) and \( x2 \) executed within a single remote desktop session, the two applications executed simultaneously \( 0.4 \times 11 \) (4) times, and sequentially \( 0.6 \times 11 \) (7) times.

Such a matrix is continuously updated in an on-line manner with aid of the dependency tracker and model updater.

Note that in our considered system, \( P_{ij}^{Sim} + P_{ij}^{Seq} = 1 \). In practice, we would hence need to store only one of these values in the matrix. The other can be inferred. However, for the purpose of illustration of the concepts in this thesis, we show both of these values within the matrix.

86
6.4.2 Dependency Tracker

The dependency tracker interfaces with a monitoring system to obtain traces of usage patterns for a remote desktop session. In practical terms, such traces would correspond to process logs for the remote desktop session. The tracker is responsible for analyzing the trace information for two kinds of dependencies Sim and Seq. We assume that each entry in the process log contains a list of processes as obtained from tools, such as `ps` (in Unix). The entries are collected at regular time intervals. Given such a process log, the tracker applies the following rules:

**Rule 1:** Two applications are said to have satisfied the Sim dependency condition if there exists one entry in process log containing both the applications executing concurrently.

**Rule 2:** Two applications are said to have satisfied the Seq dependency condition if there exists no entry in process log containing both the applications executing concurrently.

The dependency matrix is then updated as follows.

\[
\begin{align*}
P_{ij}^{Sim} &= \frac{(P_{ij}^{Sim} \ast N_{ij}) + 1}{N_{ij} + 1} & \text{if the Sim condition is true} \\
N_{ij} &= N_{ij} + 1 \\
P_{ij}^{Seq} &= \frac{(P_{ij}^{Seq} \ast N_{ij})}{N_{ij} + 1} & \text{if the Seq condition is true}
\end{align*}
\]

6.5 Performance Model Evaluation

In this section, we present algorithm to evaluate the model equations. The output of the evaluation is the resource allocation values to be made for the remote desktop session. We
describe the evaluation process below.

**Input:** The list of applications desired within a remote desktop session. \(A_i, 1 \leq i \leq n\)

**Output:** Resource Allocations for the remote desktop session.

**Step 1:** Obtain the resource shares requirements for each individual application \(A_i\), using the equations from Chapter 5.

**Step 2:** Select a dependency structure for the given set of applications. The dependency structure is one of \(Sim, Seq, Mix\). The derivation of the dependency structure to choose is as follows:

- Obtain from the dependency database the set of individual probabilities among application pairs within the set \(A = \{A_i, 1 \leq i \leq n\}\), i.e.,
  \[M[i,j]; i, j \in A\].

- Determine the probabilities that the set of applications in \(A\) would execute simultaneously and sequentially respectively. The probability that all applications in \(A\) would execute simultaneously is the probability that all of the individual application pairs, \(\{A_i, A_j\}; i, j \in A\), would execute simultaneously. The Matrix \(M\) gives us the probability values among each such application pair. We assume that there exists no functional dependency among the applications. Based on this assumption, the individual events can be taken to be independent. For example, in an application set \(\{A1, A2, A3\}\), the event \(\{A1, A2\}\) executing simultaneously is taken to be independent of the event that \(\{A2, A3\}\) execute simultaneously. A combination of functional and timing dependencies would require consideration of conditional probabilities. Exploration of such combination is planned for Future Work. Below, we only assume independent events,
and derive the probabilities as

\[
P_A^{Sim} = \prod_{i=A}^{i=A_n} \prod_{j=A}^{j=A_n} P_{ij}^{Sim}, \ i, j \in A
\]

\[
P_A^{Seq} = \prod_{i=A}^{i=A_n} \prod_{j=A}^{j=A_n} P_{ij}^{Seq}, \ i, j \in A
\]

such that

\[
P_A^{Sim} = Pr(\text{Applications in } A\text{ execute simultaneously})
\]

\[
P_A^{Seq} = Pr(\text{Applications in } A\text{ execute sequentially})
\]

- If \( P_A^{Sim} \geq T_h \) & \( P_A^{Seq} < T_h \), then the dependency structure is taken to be \( Sim \)

- If \( P_A^{Seq} \geq T_h \) & \( P_A^{Sim} < T_h \), then the dependency structure is taken to be \( Seq \)

- Else, the dependency structure is taken to be \( Mix \).

\( T_h \) represents an upper threshold value that is set by an administrator. Typical examples would be values between 0.7 and 0.9.

**Step 3:** Once the dependency structure is determined, we evaluate our performance model equation, \( ResourceShares_{desktop} = F(D,R) \).

By default, we apply our policy presented in Section 6.3 to evaluate \( F \). If the dependency structure chosen is \( Mix \), the probability values evaluated above are used to guide the percentile value to be chosen for the resource shares (based on the discussion in Section 6.3).

Once generated, the model instance is stored within the model repository.
6.6 Notation

Table 6.2: Additional Notations Introduced in This Chapter

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{desktop}$</td>
<td>Aggregate CPU resource share requirement in cycles/second for the entire remote desktop session (multiple apps)</td>
</tr>
<tr>
<td>$L_{N_{desktop}}$</td>
<td>Acceptable network latency for the remote desktop session</td>
</tr>
<tr>
<td>$L_{S_{desktop}}$</td>
<td>Acceptable storage latency for the remote desktop session</td>
</tr>
<tr>
<td>$N_{desktop}$</td>
<td>Aggregate network bandwidth requirement for the entire remote desktop session (multiple apps)</td>
</tr>
<tr>
<td>$S_{desktop}$</td>
<td>Aggregate storage bandwidth requirement for the entire remote desktop session (multiple apps)</td>
</tr>
</tbody>
</table>
Chapter 7

Resource Management of Remote Desktop Sessions

In this chapter, we present our resource management solution for desktop sessions that is driven by remote desktop models. We first describe the framework for the resource management solution. Subsequently, we describe the details of our proposed model-driven admission control and resource assignment systems. Finally, we present validation of the resource management system. The validation is presented in two parts. In the first part, we describe a prototype implementation of the resource management framework. In the second part, we present simulation analysis studies on resource sharing strategies using model-driven resource management.

7.1 Execution Flow

Chapter 4 described the conceptual view of a remote desktop utility. End-users submit requests for hosting remote desktops within this utility. These requests are handled by a Resource Management Service. The functional execution flow that follows within the

![Functional Execution Flow Diagram]

Figure 7.1: Functional Execution Flow

91
Figure 7.2: Resource Management Framework Principal Entities and Information Flow

Resource Management Service to process this request goes through four main phases (see Figure 7.1). The first phase is the request selection phase in which requests are selected from an Input Queue for processing. The second phase is the model evaluation and management phase in which application models for the requested remote desktop session are evaluated according to the model equations presented in Chapter 5 and Chapter 6. The third phase is the model-based provisioning phase wherein the requests go through admission control and resource assignment. These resource management functions are guided by the models. The final phase includes model-based adaptation, dependency discovery, and runtime admission control. The dependency discovery leads to an update of the models. Adaptation actions are of two types - local adaptation wherein the updated models are not used. Global Adaptation wherein the updated models are utilized. In this case, the Admission Control is performed once again for the application service. In this dissertation, we do not address model-based adaptation and runtime admission control.

7.2 Framework and Information Flow

The principal entities of the resource management framework are the resource management server (RMS), the compute servers, and the management overlay network (see Figure 7.2).
The functional execution flow described in the previous section is supported by the proposed framework. In this section, we describe the various sub-entities in our proposed framework along with the high level information flow. Figure 7.2, Figure 7.4, and Figure 7.6 illustrate the flow.

The information flow starts with the Application Consumer contacting the Application Provider and requesting for a remote desktop session (see Figure 7.2) (1). The Application Provider in response creates a request template for the session and submits it to the Resource Management Server (2). The requests sent by the Application Provider is received by a Request Receiver Service. We take a Service Oriented Computing approach and expose resource management API’s to the outside world. An example XML request template is shown in Figure 7.3. On receiving the request, the Request Receiver Service authenticates

```
<Request>
  <HardwareInfo>
    <ComputeServer>
      <NumCPU>1</NumCPU>
      <Type>Intel</Type>
      <OSType>Linux</OSType>
    </ComputeServer>
    <StorageServer>
      <DiskSpace>600</DiskSpace>
    </StorageServer>
  </HardwareInfo>
  <ApplicationList>
    <Application>
      <Name>RenderName</Name>
      <InputFiles>foo, xyz, abc</InputFiles>
      <AppVersion>1.2</AppVersion>
    </Application>
    <Application>
      <Name>RealPlayer</Name>
      <AppVersion>3.0</AppVersion>
    </Application>
    <Application>
      <Name>Archiver</Name>
      <AppVersion>5.0</AppVersion>
    </Application>
    ...
  </ApplicationList>
  <SessionReqt>
    <DisplayTechnology>VNC</DisplayTechnology>
    <Duration>60</Duration>
    <CompressionReqt>3</CompressionReqt>
  </SessionReqt>
  <SecurityReqt>
    <Encryption>RSA</Encryption>
    <Vpn>No</Vpn>
  </SecurityReqt>
</Request>
```

Figure 7.3: Request Template

the sender, and if successful, places the request within the Input Queue (3a) (see Figure 7.4). The requester is blocked until the application service hosting is completed, at which
Figure 7.4: Entities at the Resource Management Server and the Information Flow

point it is given a handle to the service. Optionally, the sender could be given a job id, and it can periodically check the status of his request. Both options can be supported and is an implementation choice to certain degree.

A Request Selection module picks requests from the Input Queue according to resource selection policies (3b). We propose to apply Multi-Queue Request Selection schemes with moving priorities, that have been studied within the Operating System literature towards our purpose [82]. Such schemes have traditionally been applied for local OS scheduling. In our framework, we apply the scheme in a very different context at the Resource Management Server. Our requests are those submitted by Application Providers. We associate a request selection deadline with each Application Provider category. The higher the category of the application provider, the lower is the deadline value. Multiple Queues are maintained at the server with priorities. The priorities correspond to deadlines denoted as $D_1, D_2, D_3, \ldots, D_n$ in order; $D_1 < D_2 < D_3 < \ldots < D_n$ (refer to Figure 7.5). Requests are always taken from queue with deadline $D_1$. Each request has initial credits corresponding to the deadline. Thus requests with deadline $D_n$ are assigned credits equal to $D_n$. For queues $D_3$ to $D_n$, when the value of credits become equal to the deadline corresponding to the upper queue, the requests are moved up. For example, if the credits for a request in queue $D_3$ becomes equal to $D_2$, then that request is moved up to queue $D_2$. For requests in queue $D_2$, when
Figure 7.5: Illustration of Movement of Requests within Priority Queues

credits = THRESHOLD, we move it to D1. Once the request is in queue D1, it is ready to be scheduled. THRESHOLD is a function of the average wait time of requests in queue D1. Ideally, with zero wait times of requests in queue D1, THRESHOLD would be 0. If D1 is empty for n clock ticks, a daemon pre-maturely moves some of the requests from D2. If D2 is also empty, the daemon pre-maturely move requests from D3 and so on.

Once the request is selected using the above described selection policy, the selected request is handed to a Static Requirement Filter (3c) that does a match of the Application Providers’ preference of static characteristics, e.g., hardware requirements, with the information of the resources stored in the Resource Information Models. If there are no matching resources, the process is terminated. Resource Information models in our framework are assumed to be based on Common Information Models (CIM) [83] from Distributed Management Task Force (DMTF), or Resource Description Framework (RDF) [84] from W3C. These models store the static information about available infrastructure elements - compute servers, storage servers, network elements, as well as the associated network topology information.

After the pass through Static Requirement Filter, the request is passed on to the Application Model Management System (3d). A model evaluator module evaluates the model for the requested remote desktop session. With this dynamically evaluated model output, the request is handed to a Site Admission Control System (3e), which performs the admission check for the requested desktop session. The model is used to guide the admission control
Figure 7.6: Entities at the Compute Server and the Information Flow

process. If after the admission check, no resources can be found matching the desktop’s resource requirements, the request is placed within a Pending Queue. The request is picked again later for processing when resources get freed up. If the admission check is successful, all of the candidate set of QoS-satisfying compute servers are determined.

The request is then handed to a *Resource Assignment System (3f)* which selects one of the candidate set of compute servers from among those obtained at the end of Site Admission Control. Once the selection is complete, the resource assignment tables and resource information models are updated to reflect the new assignment of the desktop session to the selected resources. With the resource assignment completed, the request is forwarded to a *Job Dispatcher Service (4)* (see Figure 7.2).

The Job Dispatcher Service communicates with *Local Resource Manager Service* on the assigned compute servers (5), and requests for a reservation to be made for the desktop session in terms of fine grained resources, such as CPU, network bandwidth, and storage bandwidth. It then waits for confirmation messages to be received from the Local Resource Manager Service. In response to the requests sent by the Job Dispatcher, the Local Resource Managers create *dynamic accounts (6a)* (see Figure 7.6). Unlike normal user accounts which remain permanently assigned to the same real-world user, a dynamic account is assigned to a user temporarily. Once the account is created, the Local Resource Manager then interfaces with *virtual machine and server partitioning technologies* like HP Process Resource
Manager (PRM) [80], and forwards to them the reservation values to create a virtual machine conforming to those values (6b). When the reservation is complete, a confirmation message is sent to the Job Dispatcher Service (7) (see Figure 7.2). The Job Dispatcher now interfaces with a Deployment Service to deploy the remote desktop display server with the requested set of applications onto the assigned compute server (8). The various Monitoring Data Collector and Delivery Agents for the remote desktop session are also started at the end of deployment. After the desktop session is activated, the Job Dispatcher Service propagates the confirmation through the various RMS entities (9) and it is passed on to the Request Receiver Service (10) who then hands it to the Application Provider (11). The Application Provider would then provide this end-point reference to the Application Consumer (12).

The Application Consumer now interacts directly with the remote desktop session (13). The user requests are received by a End-User Request Receiver Service. This service may decide to apply a runtime admission control through Session Admission Control module to determine if the end-users' request to the remote desktop session should be admitted (14) (see also Figure 7.6). If QoS violations are detected by the admission control system, the request is denied. Else, the request is accepted and forwarded to the remote desktop session.

The monitoring data is sent through a monitoring overlay network to the Resource Man-
agement Server (15). The Monitoring Overlay Network exists to process and forward the monitored application data within the utility system. It has the following modules: Monitoring Data Delivery, Monitoring Data Reporter, and Data Aggregator. Figure 7.7 illustrates the overlay network and the co-ordination model based on the producer-consumer paradigm as being defined in the Grid Monitoring Architecture [85]. Some of the components act as both producers and consumers. Within the overlay, the monitored data is aggregated by a Data Aggregator. The Data Aggregator applies aggregation to the raw resource usage data obtained from the monitoring data delivery agents. Aggregation allows compaction of data to minimize storage, application of filtering for interpreting data at various granularities, and re-organization of data for computing statistics and inferring events. The Data Aggregator then sends the aggregated data to various consumers, some examples of which are shown in Figure 7.7. At the Resource Management Server, the Application Model Management System and Adaptation module process the monitoring data among others. This may result in model updates and re-allocation decisions respectively (16).

With the high level overview provided through the execution flow and the information flow within our resource framework, we now focus the next two sections describing our solution addressing site admission control and resource assignment.

### 7.3 Site Admission Control

The Site Admission Control module performs a "what-if" analysis to determine whether an application service can be admitted into the remote desktop utility system. It utilizes the models evaluated in response to the ASP’s request. Figure 7.8 illustrates the model-based nature of the admission process. We explain our algorithm in the context of the model introduced in Chapters 5 and 6.

**Input:** Resource requirements for the remote desktop session $ResourceShares_{desktop}$ eva-
Figure 7.8: Application Model-driven Site Admission Control

uated from the model, the set of compute servers $CS_{pass1} = \{CS_i; 1 \leq i \leq n\}$ that satisfy the coarse grain static resource requirements for the ASP’s request, and the infrastructure measurement data.

Output: Set of candidate compute servers, $CS_{pass2} = \{CS_i; 1 \leq i \leq m\}$ each of which can satisfy the performance needs of the remote desktop session, without violating the performance guarantees made to existing desktop sessions.

Step 1: For each compute server $CS_i$ in $CS_{pass1}$, perform the following admission check:

\[
P \cdot (T_C - U_C) \geq C_{desktop} \\
\min((T_N - U_N) \cdot N_T, T_N \cdot N_E) \geq N_{desktop} \\
\min((T_S - U_S) \cdot S_T, T_S \cdot S_E) \geq S_{desktop}
\] (7.1)

\[NL_E \leq L_{N_{desktop}}\]
\[SL_E \leq L_{S_{desktop}}\]

The expressions on the left side of the comparison operator represent the currently available resources on the compute server and those on the right side of the comparison operator represent the QoS requirement for the remote desktop session. The admission check is thus to compare that the currently available resources on the compute server can satisfy the required values for the requested remote desktop session. Note that due to the heterogeneity
in the hardware platforms e.g., CPU, we have to normalize the values of the quantities before comparison e.g., CPU utilization is expressed in cycles/second.

**Step 2:** If the admission check above is *true* for compute server $CS_i$, add it to the set $CS_{pass2}$.

**Step 3:** If the admission check fails for all of $CS_i$ in $CS_{pass1}$, then place the request into a Pending Queue to hold it until sufficient free resources become available.

**Step 4:** Else, return the set $CS_{pass2}$.

### 7.4 Resource Assignment

The Resource Assignment system is responsible for assigning one of the compute servers which satisfy the site admission check, to the ASP’s request. Figure 7.9 shows the system view of the resource assignment system. It receives input from the Model Evaluator, and the Site Admission Control system. The output assignment is updated within the Resource Information Models. The Resource Information Model is a repository which contains, among many other objects, the relationship between the assigned applications and the resource elements. The output is also provided to the Job Dispatcher Service.

An unbounded set of ASP requests $\{R_1, R_2, R_3, \ldots\}$ for the hosting of remote desktop sessions arrive at the resource management server. The requests are processed in the order of arrival of requests, and without waiting for subsequent requests. Each such request $R_i$ is considered for resource assignment. Below, we describe more formally this process for a request $R_i$.

**Input:** (i) A request, $R_i$, for a remote desktop session, and its’ resource requirement evaluated from the model equations, $ResourceShares_{desktop}$. (ii) A set of compute servers $CS_{pass2} = \{CS_i, 1 \leq i \leq m\}$ that have satisfied the Site Admission Control test. The utilization values for each of these servers is provided by the monitoring overlay and is represented.
by \((U_C, U_N, U_S)\). We are also provided with the values of the dynamic end-to-end network and storage bandwidth and latencies as shown in Figure 7.10.

**Output:** Assignment of Request \(R_i\) to a server \(CS_i\)

**Constraint:** Fractional resources on the server need to be allocated as specified by \(Resource Shares_{desktop}\).

**Optimization:** Wait time for the requests should be minimized, and the average utilization of compute servers should be maximized.

The \textit{wait time} refers to the period of time it takes for compute server(s) to be assigned to a request, starting from the time the request is received at the Resource Management Server. Wait time is a significant metric since unlike batch job submissions, after submitting the request for remote desktop session a user typically waits for the compute server to be allocated to her. In our system, the wait time is mainly determined by three parameters: the wait time in the \textit{Input Queue}\(^1\), the wait time in the \textit{Pending Queue}\(^2\) waiting for resources to be-

\(^1\)Input Queue is the queue into which the requests are placed as they arrive into the Utility System.

\(^2\)Pending Queue is the wait queue into which requests go if all the eligible compute servers for a request do not have enough available resources to satisfy the Site Admission Control performance criterion test.
come available, and processing overhead of the admission control and assignment algorithms. More precisely, wait time for a request $i$ is

$$l_i = l_{\text{input queue}} + l_{\text{pending queue}} + \text{overhead} \quad (7.2)$$

The system utilization of a compute server is represented as an integrated metric represented by

$$U_{\text{ComputeServer}} = W_{1CS} * U_C + W_{2CS} * U_N + W_{3CS} * U_S \leq 1$$

$$\sum_i W_{iCS} \leq 1 \quad (7.3)$$

The values of weights $W_{iCS}$ are determined by the Utility Infrastructure Provider and typically based on the monetary value associated with each of the resources - Compute, Network, and Storage.

The Resource Assignment problem as described so far can be stated as an *On-line Multi-Capacity Bin Packing* problem [86, 87] wherein each request $R_i$ maps to an object of multiple
parts - \( C_{\text{desktop}}, N_{\text{desktop}}, S_{\text{desktop}} \). Each compute server maps to a bin of multiple compartments - compute, network bandwidth, and storage bandwidth. The problem is to pack the objects into the bins such that the number of bins is reduced. This problem is known to be \( NP\)-hard. We therefore explore heuristic solutions to the problem.

The size of the compartments in each bin can be bounded to be of unit size by choosing utilizations as the metric of allocation. To be able to express the parts of the object as desired utilizations in a similar manner, we would have to assume that the maximum absolute capacity of the resources on the compute servers are equal. However, this will not be true given our heterogeneity assumption. We hence consider the values of the object parts - \( C_{\text{desktop}}, N_{\text{desktop}}, S_{\text{desktop}} \) to be unbounded (bounded only by physical limits). We introduce a Fitting Metric for compute server \( S_i \) that determines how well a request fits within a compute server \( S_i \)

\[
M_{\text{fitting}_i} = W_1 f * \frac{C_{\text{desktop}}}{C_{\text{available}_i}} + W_2 f * \frac{N_{\text{desktop}}}{N_{\text{available}_i}} + W_3 f * \frac{S_{\text{desktop}}}{S_{\text{available}_i}} \leq 1 \\
\sum_i W_i f \leq 1
\]  

\( i = 1 \)

\(
C_{\text{available}} = P * (T_C - U_C) \\
N_{\text{available}} = \min((T_N - U_N) * N_T, T_N * N_E) \\
S_{\text{available}} = \min((T_S - U_S) * S_T, T_S * S_E)
\)  

\( i = 1 \)

Given such a formulation, we run our Resource Assignment heuristic that finds the Best Fit for the requests. Figure 7.11 illustrates the mapping. For each of the compute servers in \( CS_{\text{pass2}} \), we evaluate the Fitting Metric. The Best Fit criterion is

\[
\max_{i=1}^m M_{\text{fitting}_i}
\]  

\( i = 1 \)

103
The running time of the best fit algorithm for each request would be $O(m)$. Assuming $n$ requests, the total running time for fitting all $n$ requests would be $O(m \times n)$.

The weights $Wi_f$ are chosen so as to optimize the number of bins used (so as to minimize the wait time for requests), and minimize the fragmentation (so that the utilization metric in Equation 7.3 is maximized for the utilized compute servers). Examples of weight assignment that takes a mean value across the various resources is to assign 1/3 for each $Wi_f$. To select the maximum, we set the weight corresponding to the term

$$\max\left(\frac{C_{desktop}}{C_{available}}, \frac{N_{desktop}}{N_{available}}, \frac{S_{desktop}}{S_{available}}\right)$$

to be equal to 1, and the others as zero. The relative values for the weight assignment are also dependent on the nature of the applications. For example, for CAD design sessions, the CPU would be the bottleneck resource variable. Similarly for financial transaction applications,
the storage bandwidth would be the bottleneck resource variable, and for office applications, the network latency would be the bottleneck resource variable.

### 7.5 Prototype Implementation

Figure 7.12 shows the testbed setup for a remote desktop utility. The Utility Infrastructure Provider consists of compute servers, data server, resource management server, and site security server. All of the nodes are Intel x86 machines running Red Hat Linux 7.3. GPDK [88] is used to provide the web portal to the end-user for submitting job requests. The application consumer submits the requests from a submission node through a web browser. Globus Toolkit 2.0 [89] is used as the middleware platform, and VNC [21] as the remote display technology. For convenience purposes, we assume that the VNC server and the applications of interest are pre-deployed onto all the compute servers. In practice, the deployment would
occur on-demand when given a request from the application provider. We have extended the functionality of Globus, so that it can also be used for submitting requests for remote desktop sessions. Globus GSI certificates are used for authentication. The Resource Allocator component at the Resource Management Server implements the Site Admission Control and the Resource Assignment modules. The more sophisticated functionalities of the resource management system is not implemented in the testbed. However, we have implemented most of these sophisticated functionalities within a simulator described in Section 7.6.

Application consumers submit job requests to the portal node specifying the list of applications desired by the user. Optionally, the user may also specify information about the static characteristics of the resource that she is seeking. If the user does not specify these requirements, the application provider specifies them based on prior agreements made with the end-user. During our demonstration, the performance model output is hand-drafted and provided as input to the resource management server. In real deployments, the performance models output would be dynamically evaluated at the Resource Management Server. However, for the sake of simplicity of implementation, we demonstrated it as a part of the request to the Resource Management Server. The portal node uses MyProxy Server to store users’ security credentials. Given a job request, the portal node uses the users’ security credentials to submit the request to the Resource Management Server. Requests at the RMS are processed by a FCFS policy. An LDAP Server stores the information models. The Site Admission Control and Resource Assignment modules make resource allocation decisions based on information from the LDAP Server. We do not have support for collecting real time utilization data of the compute servers within the testbed. Instead, we maintain an in-memory utilization table at the resource management server. Whenever a resource assignment is made, the table is updated to reflect the resource guarantees offered to the request.

Once a compute server and the fractional resource shares is assigned for the users’ request, the appropriate account pool for the session is determined. A dynamic account from this pool
is then allocated for the session. The Local Resource Manager subsequently starts a VNC server and job monitoring agents. In our implementation, we maintain an end-user session through the Portal Node. The details of the VNC server instance are returned to the Portal node as a callback, which in turn is returned to the users’ web client. The applet version of VNC client then executes within the web browser at the submission node, and it establishes a VNC session to the VNC server on the allocated compute server. On a successful VNC authentication, the user is presented with a controlled KDE Desktop environment containing only the applications and menus the user is allowed to access. The KDE desktop environment is pre-configured by the system administrator for each pool of accounts.

The session starts with default startup applications, including an interactive shell (GISH) [22]. GISH is a controlled shell that provides access control capabilities. The GISH shell has been implemented as an extension to the popular GNU bash shell for Linux and Windows. The shell source code was modified so as to include the access control modules. GISH currently checks for list of allowed executables from a file, before executing commands. The job monitoring agents started at the beginning of the session run with super-user privileges. They record the session and system information and store them in pre-determined files. The job monitoring agents check for the usage time for the session, and number of spawned processes. The system policy files contain information about the session usage time, number of allowed processes etc. along with their maximum allowed values, and actions to be taken on violation of these policies. The current default action is to KILL all the processes and end the session, on violation of the session policies.

The prototype implementation shows the feasibility of the resource management solution.

7.6 Simulation Analysis

We have built a simulation tool that implements the model-driven resource management functionalities. The tool accepts requests for hosting applications within the utility system.
The tool was primarily created to simulate requests for remote desktop sessions. However, the tool can also be used to simulate requests for other applications, as will be shown shortly. Each compute server is modeled as having two network interfaces - one for the display traffic for interactive sessions to the end-user’s thin client, and the other for storage traffic to data servers. We also model the end-to-end network bandwidth and latency between the compute server and the end-user submission nodes, as well as the end-to-end storage bandwidth and latency between the compute servers and the data servers. The requests are assumed to be picked from the Input Queues as First Come First Served (FCFS) semantics with no priorities. The simulation tool has support for assigning requests using the Least Loaded algorithm or Multi-Variable Best Fit algorithm. Our simulator is implemented in Java. The static information about the data center resources are stored in an LDAP directory, and the dynamic end-to-end information is stored as an in-memory table. We use pools of worker threads to parallelize the scheduling tasks within the simulator.

The purpose of building the simulation tool was to aid IT managers in answering key questions, such as capacity planning exercise, evaluating resource management algorithms, and/or derive best practices for configurations parameters, e.g., weight assignments in the resource assignment system. With the primary purpose of the tool being for IT managers, we present in this section certain experiment scenarios we executed using the tool that further validates the benefit of a model-driven approach.

**Experiment Scenario**

We consider a mix of batch jobs and remote desktop sessions that need to be hosted in a utility. An over-provisioning strategy would maintain separate pools of resources for the two types of jobs. With a model-driven approach, the resource management system has confidence in the amount of resources that would be needed for each request. Hence, the allocation made for each request would be equal to that evaluated by the model. As a result, the pools of resources could afford to spare idle computing cycles for a job of the
other type to be executed without impacting the application QoS guarantees. This would help in improving overall system efficiency, as a smaller number of machines could now be used to satisfy both the types of requests. Specifically, the gain in the number of machines reduced could be 100% if only a single pool of resources could satisfy both types of requests. However, this gain could come at a tradeoff. Application requests may now need to wait for machines to get assigned if all servers become busy. Otherwise, the throughput in terms of jobs finished could get affected due to sharing (especially if flexible sharing is allowed). In this section, we present results of simulation studies we did to better understand this tradeoff. We use traces of real-world use cases for the simulations.

We consider four classes of requests - Heavy Batch Job, Light Batch Job, Heavy Remote Desktop Session, Light Remote Desktop Session. Figure 7.6 shows examples of these classes of requests. We assume a workload distribution to the utility system based on time-of-day characteristics. Accordingly, we assume that during day time, the set of requests consists predominantly of Heavy Remote Desktop Session and Light Batch Job requests. During the night time, the set of requests consists predominantly of Light Remote Desktop Session and Heavy Batch Job requests.

The site admission control system implementation for a remote desktop session request checks for performance criterion described in Section 7.3. For a batch request, we check if there is a minimum required threshold CPU utilization available on a compute server. During the simulation, the CPU utilization allocated to a batch request is guaranteed to be the minimum threshold, and is allowed to exceed the minimum threshold only in case of available CPU cycles. The resource utilizations for the remote desktop sessions are always guaranteed to be equal to that of the value decided through the modeling of the remote desktop session.
Table 7.1: Examples of the Request Types

<table>
<thead>
<tr>
<th>Request type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heavy Remote Desktop Session</td>
<td>CPU intensive interactive</td>
</tr>
<tr>
<td></td>
<td>CAD design application</td>
</tr>
<tr>
<td>Light Remote Desktop Session</td>
<td>Office applications</td>
</tr>
<tr>
<td>Heavy Batch Job</td>
<td>Heavy weight structural analysis simulations</td>
</tr>
<tr>
<td>Light Batch Job</td>
<td>Compiling programs</td>
</tr>
</tbody>
</table>

Metric Description

We choose *throughput (finish time)* and *wait time* to be the performance metrics of interest. Throughput is the time taken for all of a set of requests to finish execution. The wait time refers to the period of time it takes for compute server(s) to be assigned to a request, starting from the time the request is received at the Resource Management Server. Wait time is a significant metric since unlike batch job submissions, a user after submitting the request for remote desktop session typically waits for the compute server to be allocated to her immediately. In our system, the wait time is mainly determined by three parameters: the wait time in the *Input Queue*\(^3\), the wait time in the *Pending Queue*\(^4\) waiting for resources to become available, and processing overhead of the admission control and assignment algorithms. More precisely, wait time for a request \(i\) is

\[
l_i = l_{\text{input queue}} + l_{\text{pending queue}} + \text{overhead}
\]

In our simulation results described below, there was no wait time for the requests in the Input Queue, i.e, \(l_{\text{input queue}}\) is zero, and we ignore the processing overhead of the admission control and resource assignment algorithms. Hence, the wait time is effectively determined by \(l_{\text{pending queue}}\).

---

\(^3\)Input Queue is the queue into which the requests are placed as they arrive into the Utility System.

\(^4\)Pending Queue is the wait queue into which requests go if all the eligible compute servers for a request do not have enough available resources to satisfy the Site Admission Control performance criterion test.
Table 7.2: Batch Job and Remote Desktop Session System Allocations Obtained Through Evaluation of Model Equations

<table>
<thead>
<tr>
<th>Request type</th>
<th>CPU Utilization</th>
<th>End-to-end network b/w for display traffic</th>
<th>End-to-end storage b /w</th>
<th>Duration in wall clock time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heavy Remote Desktop Session</td>
<td>15% guaranteed</td>
<td>15 Mbps</td>
<td>150 Mbps</td>
<td>6 hours</td>
</tr>
<tr>
<td></td>
<td>on a 2 GHz machine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Remote Desktop Session</td>
<td>10% guaranteed</td>
<td>10 Mbps</td>
<td>100 Mbps</td>
<td>1 hour</td>
</tr>
<tr>
<td></td>
<td>on a 2 GHz machine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heavy Batch Job</td>
<td>Minimum threshold of 35 % on a 2 GHz machine</td>
<td>0 Mbps</td>
<td>300 Mbps</td>
<td>4 hours at 35 % CPU Utilization on a 2 GHz machine</td>
</tr>
<tr>
<td>Light Batch Job</td>
<td>Minimum threshold of 5 % on a 2 GHz machine</td>
<td>0 Mbps</td>
<td>100 Mbps</td>
<td>3 hours at 5 % CPU Utilization on a 2 GHz machine</td>
</tr>
</tbody>
</table>

Experimental Setup

The experiments were conducted for a data center of size 100 compute servers. Each of the compute servers has a 2GHz processor speed, 100 Mbps network interconnect for display traffic, and a 1 Gbps interconnect for storage traffic. The dynamic end-to-end bandwidth to the end-user locations for the display traffic varies from 50-100Mbps, and the latency varies from 10-40 units. The dynamic end-to-end storage bandwidth to the file servers is 500 Mbps, and the latency varies from 5-20 units. We conducted two sets of experiments - one for the set of requests for day time activity, and the other for the set of requests for night time activity. In this set of simulations, we assume that each request is always assigned resources on only a single compute server. The system allocation values obtained through evaluations of performance model equations is shown in Table 7.6.

The system allocation values that are obtained from performance model equations for remote desktop sessions in Table 7.6 are the aggregate values as discussed in the remote desktop session model in Chapter 6. The acceptable latencies are greater than the available end-to-end latency values. As mentioned earlier in the section, the day time experiments consists predominantly of Heavy Remote Desktop Session and Light Batch Job requests. During the night time, the set of requests consist predominantly of Light Remote Desktop
Table 7.3: Request Description

<table>
<thead>
<tr>
<th>Experiment Type</th>
<th>Remote Desktop Interactive Session Requests</th>
<th>Batch Job Requests</th>
<th>Arrival Rate for Interactive Session Requests</th>
<th>Arrival Rate for Batch Job Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day Time Experiment</td>
<td>Heavy Remote Desktop Session requests only</td>
<td>Light Batch Job requests only</td>
<td>Poisson distribution; last request arrives at 6 hours into the experiment</td>
<td>Poisson distribution; requests arrive throughout the 12 hour experiment</td>
</tr>
<tr>
<td>(12 hours)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Night Time Experiment</td>
<td>Light Remote Desktop Session requests only</td>
<td>Heavy Batch Job requests only</td>
<td>Poisson distribution; requests arrive throughout the 12 hour experiment</td>
<td>All requests arrive in a batch at the beginning of experiment (Bursty arrival at time 0 of the experiment)</td>
</tr>
<tr>
<td>(12 hours)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.4: Results for Day Time Experiments
No Resource Sharing among Mixed Workloads
Number of Dedicated nodes for each job type = 100

<table>
<thead>
<tr>
<th>Metric</th>
<th>$N_{Batch} = 100$</th>
<th>$N_{Batch} = 200$</th>
<th>$N_{Desktop} = 500$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput (Finish Time in minutes)</td>
<td>728</td>
<td>730</td>
<td>716</td>
</tr>
<tr>
<td>Max Wait Time in minutes</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Complete Resource Sharing among Mixed Workloads
Number of Shared nodes = 100

<table>
<thead>
<tr>
<th>Metric</th>
<th>$N_{Batch} = 100$ and $N_{Desktop} = 500$</th>
<th>$N_{Batch} = 200$ and $N_{Desktop} = 500$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput (Finish Time in minutes)</td>
<td>728 (Batch Jobs) 722 (Interactive Sessions)</td>
<td>730 (Batch Jobs) 724 (Interactive Sessions)</td>
</tr>
<tr>
<td>Max Wait Time in minutes</td>
<td>6</td>
<td>11</td>
</tr>
</tbody>
</table>

$N_{Batch}$ = Number of Batch Jobs (of type Light Batch)
$N_{Desktop}$ = Number of Desktop Sessions (of type Heavy Remote Desktop)
Session and Heavy Batch Job requests. Based on this workload distribution, we generate separate 12 hour synthetic requests for the day and night time as shown in Figure 7.6. Note that these are requests given to the Resource Management Server.

Results

We first describe the results for the Day Time experiment which consisted of 500 Heavy Remote Desktop Session requests. We varied the number of Light Batch Job requests in the system. Figure 7.4 shows the finish time (Throughput) of the system with 100 and 200 Light Batch Job requests and 500 Heavy Remote Desktop Session requests. We see that the finish time for the Light Batch jobs is unaffected when the two sets of requests are combined on the same set of compute nodes. The finish time for the Heavy Remote Desktop sessions degrades by 0.8% and 1.1% with the addition of 100 and 200 Light Batch jobs in the system respectively compared to executing the Heavy Remote Desktop Session requests on a separate set of compute nodes with no Light Batch jobs. As a result, the finish time goes up by only 6 and 8 minutes respectively and is only slightly above 12 hours. Figure 7.4 also shows the maximum wait time in the Pending Queue for the requests with 100 and 200 Light Batch Job requests in addition to the 500 ‘Heavy Remote Desktop Session’ requests. We see that the maximum Wait time for the requests increases from 6 to 11 minutes with addition of 100 and 200 Light Batch Job requests. Thus, with 100 additional requests of Light Batch jobs, a client asking for a remote desktop interactive session may have to wait 6 minutes before being allocated a resource for interactive use. We also ran additional experiments and increased the number of Light Batch Jobs in the system to 2000 running 500 Heavy Remote Desktop Sessions. We found that even with 2000 Light Batch Jobs in the system, the finish time for Heavy Remote Desktop Sessions degrades by 8.9% (finish time is

\[\text{in the simulations, there was no wait time for the requests in the Input Queue and we ignore the processing overhead of the admission control and resource assignment algorithms. Hence, the wait time presented here is that incurred if the requests do not satisfy the Site Admission Control test and hence go to the Pending Queue.}\]
Table 7.5: Results for Night Time Experiments
No Resource Sharing among Mixed Workloads
Number of Dedicated nodes for each job type = 100

<table>
<thead>
<tr>
<th>Metric</th>
<th>$N_{Desktop} = 30$</th>
<th>$N_{Desktop} = 200$</th>
<th>$N_{Batch} = 500$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput (Finish Time</td>
<td>65</td>
<td>779</td>
<td>622</td>
</tr>
<tr>
<td>in minutes)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complete Resource Sharing among Mixed Workloads
Number of Shared nodes = 100

<table>
<thead>
<tr>
<th>Metric</th>
<th>$N_{Desktop} = 30$ and $N_{Batch} = 500$</th>
<th>$N_{Desktop} = 200$ and $N_{Batch} = 500$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput (Finish Time</td>
<td>65 (Interactive Session)</td>
<td>779 (Interactive Sessions)</td>
</tr>
<tr>
<td>in minutes)</td>
<td>660 (Batch Jobs)</td>
<td></td>
</tr>
</tbody>
</table>

$N_{Batch} =$ Number of Batch Jobs (of type Heavy Batch)
$N_{Desktop} =$ Number of Desktop Sessions (of type Light Remote Desktop)

780 minutes). However, the maximum wait time for the requests goes to 67 minutes. The maximum wait time starts to degrade beyond 20 minutes with greater than 700 Light Batch Jobs. The 95 percentile value for the Wait Time was found to be 0 for upto 700 Light Batch Jobs and 54 minutes for 2000 Light Batch Jobs. With the addition of upto 700 Light Batch jobs, the number of requests that waited in the Pending Queue was thus less than 5% of the requests. A Site Policy decision could restrict the allowed number of Light Batch Jobs in the system taking into consideration the acceptable degradation in the waiting time and finish time for the requests.

We now describe the results for the Night Time experiment which consisted of 500 Heavy Batch Job requests. We varied the number of Light Remote Desktop Session requests in the system. Tables 7.5 shows the finish time (Throughput) of the system with 30 and 200 Light Remote Desktop Session requests and 500 Heavy Batch Job requests. We see that the finish time for the Light Remote Desktop sessions is unaffected when the two requests are combined on the same set of compute nodes. The finish time for the Heavy Batch jobs degrades by 6.1% and 10.6% with the addition of 30 and 200 Light Remote Desktop sessions.
in the system respectively, compared to executing the request on a separate set of compute nodes with no Light Remote Desktop sessions. As a result, the finish time goes up by 38 and 66 minutes respectively. We expect the number of Light Remote Desktop sessions in the system to be fairly low during night time and do not expect it to go much beyond 30 sessions. The case for 200 Light Remote Desktop sessions thus represents a very extreme case, however even then we see that the finish time for the Heavy Batch jobs is within acceptable limits and the finish time for the system is still within the 12 hour period. We also ran additional experiments and increased the number of Light Remote Desktop Sessions in the system to 2000, while keeping the number of Heavy Batch Jobs in the system to 500. We found that the finish time for the Heavy Batch Jobs with 2000 Light Remote Desktop Sessions degrades by 15.75% (finish time is 720 minutes).

**Evaluation of Results**

The results show that the degradation in terms of throughput and wait time is not significant for the experiment scenario that we consider. This implies that the sharing of resources enabled through a model-driven approach can save costs equal to a complete pool of resources, while at the same guaranteeing application QoS performance, and without much degradation for throughput and wait time. The results are very much dependent on the application mix, the workload distribution, the QoS requirements, and the performance models. For the experiments that we conducted, we choose the values of these parameters based on corresponding real-world use cases. A different set of chosen values would have given a different result. But since we are most interested in real-world scenarios that we emulated in our experiments, the results are very encouraging. More importantly, it is another demonstration of the benefits a model-driven approach can bring to resource management.
### 7.7 Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_E$</td>
<td>Dynamic End-to-end network bandwidth between the compute server and the users’ thin client (for remote display traffic)</td>
</tr>
<tr>
<td>$NL_E$</td>
<td>End-to-end network latency between the compute server and the users’ thin client</td>
</tr>
<tr>
<td>$N_T$</td>
<td>Total network bandwidth for the compute server</td>
</tr>
<tr>
<td>$P$</td>
<td>CPU Processor speed</td>
</tr>
<tr>
<td>$S_E$</td>
<td>Dynamic End-to-end storage bandwidth between the compute server and the file server hosting the users’ data (for remote storage traffic)</td>
</tr>
<tr>
<td>$SL_E$</td>
<td>End-to-end storage latency between the compute server and the file server hosting the users’ data</td>
</tr>
<tr>
<td>$S_T$</td>
<td>Total storage bandwidth for the compute server</td>
</tr>
<tr>
<td>$T_C$</td>
<td>Maximum Threshold percentage set for the CPU Utilization on the compute server eg. 80%</td>
</tr>
<tr>
<td>$T_N$</td>
<td>Max Threshold percentage set for the Network Utilization on the compute server</td>
</tr>
<tr>
<td>$T_S$</td>
<td>Max Threshold percentage set for the Storage Utilization on the compute server</td>
</tr>
<tr>
<td>$U_C$</td>
<td>Current total CPU percentage being utilized on the compute server</td>
</tr>
<tr>
<td>$U_N$</td>
<td>Current total network bandwidth percentage being utilized on the compute server</td>
</tr>
<tr>
<td>$U_S$</td>
<td>Current total storage bandwidth percentage being utilized on the compute server</td>
</tr>
</tbody>
</table>
Chapter 8

Related Work

In this chapter, we review related work. We classify related work into the following categories: Computing and Storage Systems, Resource Management Systems, Service Oriented Architecture.

8.1 Computing and Storage Systems

We review in this section background knowledge of various computing and storage systems.

8.1.1 Traditional Scientific Grids

Several scientific grids exist, eg. NASA’s Information Power Grid (IPG) [90], Tera Grid [91], Euro Grid [92]; West Grid [10], China Grid [9]. Example applications for these Scientific Grids are computational fluid dynamics (CFD) jobs, launch vehicle designs, drug analysis, large-scale data analysis, weather forecasting, biomolecular computing, rocket simulation, cosmology, quantum chemistry, space-time meshes, structural dynamics etc. These applications are typically batch jobs, requiring access to global resources and datasets. Globus [89] is the Grid software used by a majority of todays’ Scientific Grids. The motivation for Scientific Grids is due in part to the increasing computation power needed to solve science problems, the vast size of datasets that the scientific investigations produce, and the need for collaboration among scientists on a global scale. For example, a high energy physics collaboration that shared and analyzed data from the D0 experiment spanned 73 institutions in 18 countries, with thousands of scientists involved, of which hundreds accessed its
resources simultaneously. During the first half of 2002, about 300 D0 users submitted 2.7 million requests and retrieved 824 TB of data [93].

In our research, we look at the use of Grids beyond traditional scientific applications. We investigate Commercial Grids hosting applications of interest to enterprises, and provided as a utility by Service Providers.

8.1.2 Planetlab

Planetlab [11, 12] is a research testbed consisting of nodes distributed across the globe. As of February 2006, it consists of 648 machines, hosted by 305 sites, spanning over 25 countries. A majority of the machines on Planetlab are hosted by research institutions. The goal is for Planetlab to grow to 1,000 widely distributed nodes that peer with the majority of the Internet’s regional and long-haul backbones. All Planetlab machines run a common software package that includes a Linux-based operating system; mechanisms for bootstrapping nodes and distributing software updates; and a facility for managing user accounts and distributing keys. The key objective of the software is to support distributed virtualization—the ability to allocate a slice of PlanetLab’s network-wide hardware resources to an application. This allows an application to run across all (or some) of the machines distributed over the globe, where at any given time, multiple applications may be running in different slices of PlanetLab. Planetlab is also an overlay network testbed. Research groups are able to request a PlanetLab slice: a set of nodes on which the service receives a virtual machine (VM). Within a slice, the users can experiment with a variety of planetary-scale services, including file sharing and network-embedded storage, content distribution networks, routing and multicast overlays, QoS overlays, scalable object location, scalable event propagation, anomaly detection mechanisms, and network measurement tools. There are currently over 275 active research projects running on PlanetLab.

In our research, we view Planetlab as an example of utility system.
8.1.3 Clusters

Numerous efforts have been made in the area of clusters. We only highlight a few interesting examples.

**NOW.** The Berkeley NOW [94] consists of a cluster of commodity workstations with high-speed interconnects. NOW provides a global software and hardware layer to transparently support both parallel and sequential workloads. NOW applications can leverage Active Messages, a scalable and serverless file systems, and has support for scheduling and job support.

**Google.** The Google Cluster Architecture [95] consists of commodity-class PCs with fault-tolerant software to handle the workload for a web search application. The web search application affords easy parallelization: Different queries run on different processors, and the overall index is partitioned so that a single query can use multiple processors. Google’s application is relatively homogeneous, where most servers run one of very few applications. Google uses replication for fault-tolerance, and load balancing for better system utilization.

**Web Server Farms.** A server farm is a group of networked servers that are housed in one location. A server farm streamlines internal processes by distributing the workload between the individual components of the farm and expedites computing processes by harnessing the power of multiple servers. The farms rely on load-balancing software that accomplishes such tasks as tracking demand for processing power from different machines, prioritizing the tasks and scheduling and rescheduling them depending on priority and demand that users put on the network. When one server in the farm fails, another can step in as a backup. A Web server farm, or Web farm, refers to either a Web site that runs off of more than one server or an ISP that provides Web hosting services using multiple servers. There are several web hosting services today [96] that use Web farms.

In contrast to existing cluster systems, we investigate in our research a data center environment consisting of resources that are heterogeneous, globally distributed, host diverse
applications, support virtualization, qualities of service, and autonomy. Further, each server hosts an individual commodity operating system.

8.1.4 Thin Client/Remote Desktop Computing Systems

Thin Client Computing is an emerging paradigm. Commercial products in this space primarily fall into three categories. (i) Thin client hardware, e.g., HP Thin Client Systems [97], Wyse Technologies [98], Neoware Systems [99]. (ii) Remote display technologies, e.g., Citrix MetaFrame [18], Microsoft Terminal Services [20], SunRay Systems[100], VNC [21]. (iii) Complete solutions that bundle the remote display technology with hardware, e.g., HP's Consolidated Client Infrastructure [101], HP’s Server Based Computing systems [102] and ClearCube [103].

There are research efforts as well in this space. Jason Nieh et. al have conducted performance comparisons of various thin client systems [104]. They have also proposed a new remote display architecture called THINC [105]. Research efforts have been made to build complete desktop systems such as SLIM [106] and Collective [107]. Studies have been conducted to determine appropriate metrics and their measurements for interactive workloads [108, 79, 109]. Microsoft has undertaken capacity management exercises for Microsoft Terminal Services with NEC and HP. They conduct controlled execution of scripts for different categories of users - knowledge worker, high performance worker etc., and measure response times. An estimate is then made for capacity planning [110]. However, it does not accurately address virtualization, multiple resources, heterogeneous workloads (multiple applications), long term effects.

In our work, we demonstrate our model-driven methodology for a remote desktop utility. Our work is focused on modeling the performance requirements of remote desktop sessions and applying it to resource management in a utility system.
8.2 Resource Management Systems

In this section, we review related work on resource management systems. We classify existing work by application domain and by approach. We end the section with a comparison on how our work differs.

8.2.1 Classification By Application Domain

In here, we look at the application domains - high performance computing, multimedia applications, web servers, 3-tier systems, web services.

High performance computing

Numerous efforts have been made in the scheduling area for high performance computing. We describe below a few representative ones.

Condor [111] is a specialized workload management system for compute-intensive batch jobs. It provides job queuing mechanism, scheduling policy, priority scheme, and resource monitoring. In addition to the capability of managing dedicated compute nodes, Condor can also effectively harness wasted CPU power from otherwise idle desktop workstations. If the machine is no longer available (such as a key press detected), Condor transparently checkpoints and migrates the job to a different machine which would otherwise be idle. Condor is also able to transparently redirect all of a job’s I/O requests back to the submitting machine. It also provides a ClassAd mechanism for matching resource requests (jobs) with resource offers (machines), and can also be used to build Grid-style computing environments that cross administrative boundaries.

Legion [112] is an object-based, meta-systems software project at the University of Virginia with a vision to create a worldwide virtual computer. The Legion system addresses key issues such as scalability, programming ease, fault tolerance, security, site autonomy, etc. Legion is designed to support large degrees of parallelism in application code and manage

121
the complexities of the physical system for the user. In Legion, resource management is invoked not just for running jobs, but also to place other Grid components, such as files, directories, databases etc. Placement is the key component. Legion provides a scheduling framework rather than scheduling algorithms. Legion, unfortunately, has not had as much success as envisioned. The Legion members state in [113] the lessons learned: people are reluctant to write their own scheduling algorithm, the scheduler writers also had to know too much about Legion internals, and that over-complex schedulers are unnecessary.

Maui [114] is an advanced job scheduler for use on clusters and supercomputers. It is a highly optimized and configurable tool capable of supporting a large array of scheduling policies, dynamic priorities, extensive reservations, and fairshare. It improves the manageability and efficiency of machines ranging from clusters of a few processors to multi-teraflp supercomputers. Some of the interesting features provided by Maui are: extensive job priority policies and configurations, multi-resource admin and job advance reservation support, multiple configurable backfill policies, advanced built-in HPC simulator for analyzing workload, resource, and policy changes. Maui interfaces with numerous resource management systems supporting many scheduling API’s. Moab Grid Scheduler (previously codenamed Silver) [115] is an optimizing advanced reservation based grid scheduler which allows distributed workload to be run across independent clusters. It may be used either with or without Globus and provides integrated account and data management services.

Platform LSF [116] is software for managing and accelerating batch workload processing for compute-and data-intensive applications. It fully utilizes all IT resources regardless of operating system, including desktops, servers and mainframes. It is based on the production-proven, open, grid-enabling, Virtual Execution Machine (VEM) architecture, and supports a set of intelligent scheduling policies: fairshare, preemption, advance reservation, resource reservation. Platform LSF MultiCluster [117] extends an organization’s reach to share virtualized resources beyond a single Platform LSF cluster to span geographical locations. With Platform LSF MultiCluster, local ownership and control is maintained ensuring priority
access to any local cluster while providing global access across an enterprise grid. LSF MultiCluster supports distributed resource management software organization.

The Portable Batch System, PBS Pro [118] is a batch workload management solution for High Performance Computing systems and Linux clusters. The batch system allows a site to define and implement policy as to what types of resources and how much of each resource can be used by different jobs. The batch system also provides a mechanism with which a user can insure a job will have access to the resources required to complete. Clients do not create or modify objects directly, but depend upon the server which manages those objects. A batch server is a persistent process or set of processes, such as a daemon. It provides batch services like creating, routing, executing, modifying, or deleting jobs for batch clients. A batch server may at times request services of other servers. User, operator, and administrator commands are batch clients. Some of the key features of PBS Pro are cross-system scheduling, server failover and redundancy, automatic file staging, advance reservations, distributed clustering.

Globus Resource Management. The Globus Toolkit includes a set of service components collectively referred to as the Grid Resource Allocation and Management (GRAM). GRAM simplifies the use of remote systems by providing a single standard interface for requesting and using remote system resources for the execution of "jobs". The most common use (and the best supported use) of GRAM is remote job submission and control. GRAM does not provide scheduling or resource brokering capabilities. A wide variety of meta schedulers and resource brokers that leverage GRAM mechanisms have been developed. GARA [119] is the Globus Architecture for Reservation and Allocation that provides support for advance reservations across heterogeneous collections of shared resources. GARA exposes both reservations and objects as first-class, abstract objects; defines uniform representations and operations for diverse resource types; and uses an information service to reveal site-specific policies. These constructs enable the construction of reusable co-reservation and co-allocation agents that can combine domain- and resource specific knowledge to discover, reserve, and allocate resources that meet application QoS requirements. A prototype GARA
implementation has been built that supports three different resource types - parallel computers, individual CPUs under control of the Dynamic Soft Real-time Scheduler, and Integrated Services networks.

**Web Server Systems**

Web Server load balancing is used in web hosting sites. Several web servers are deployed that can each service a user’s request. A load balancing system determines the least loaded web server, and directs the request to that server. The load balancer is typically implemented as a hardware switching device.

Efforts have been made in the past to provide QoS support in web server systems. A session based admission control system is described in [120]. The proposed system prevents a web server from becoming overloaded and ensures that longer sessions can be completed. A session in this work is defined as a sequence of web client’s individual requests. The proposed system specifically addresses only the requirements of web requests. Measurement based admission control systems have been studied in [121, 122, 123]. In such systems, real-time measurement values are used for the admission process.

A proposal for a new facility for resource management in server systems has been made in [7]. They propose and evaluate a new operating system abstraction called a resource container, which separates the notion of a protection domain from that of a resource principal. Resource containers enable fine-grained resource management in server systems and allow the development of robust servers, with simple and firm control over priority policies.

**Multimedia Applications**

Efforts have also been made within the area of admission control in multimedia servers. A statistical admission control algorithm is presented in [124] which exploits the variation in access times of media blocks from disk as well as the variation in client load induced by variable rate compression schemes, and provides statistical service guarantees to each
client. An approach to admit whole multimedia sessions instead of single media streams is described in [67]. The resource demands within a session are correlated to user behavior which is modeled as Continuous Time Markov Chains (CTMCs).

Qualman [64] is a QoS-aware resource management model for distributed multimedia applications. The proposed resource model for shared resources incorporates the resource scheduler, and the resource broker, which provides negotiation, admission and reservation capabilities for sharing resources such as CPU, network, or memory corresponding to requested QoS. Combined with the resource schedulers, they provide a more predictable and finer granularity control of resources to the applications during the end-to-end multimedia communication. QualMan consists of CPU, memory, and communication servers, which are deployed as a loadable middleware. Each of these servers consist of a broker and a corresponding resource controller.

**Multi-Tier Systems**

Recent efforts have been made towards resource management of enterprise multi-tier systems. Techniques that support advance resource reservation and admission control for business e-commerce workloads are presented in [125]. The resource demands of such applications are characterized statistically using application demand profiles. An admission control system exploits the profiles to enable the overbooking of resources while offering statistical assurances regarding access to resources. Different assurance levels correspond to alternative classes of service.

Oceano is a prototype of a highly available, scalable, and manageable infrastructure for an e-business computing utility [126]. The hosting environment is divided into secure domains, each supporting one customer. These domains are dynamic: the resources assigned to them may be augmented when load increases and reduced when load dips. This dynamic resource allocation enables flexible SLAs with customers in an environment where peak loads are an order of magnitude greater than the normal steady state. The resource management
functions provided are bandwidth management, server management, application and data management.

A technique for dynamic provisioning for multi-tier Internet applications is presented in [127]. The technique employs a flexible queuing model to determine how much resources to allocate to each tier of the application, and a combination of predictive and reactive methods that determine when to provision these resource, both at large and small time scales.

Web Services

Resource Management for web services is an emerging area. We summarize below two very recent works in this area.

In [128], a performance management system for cluster-based web services is presented. The system supports multiple classes of web services traffic and allocates server resources dynamically so as to maximize the expected value of a given cluster utility function in the face of fluctuating loads. The utility function is a function of the performance (response time) delivered to the various classes. There are two nested levels of management – the inner level centers on queuing and scheduling of request messages; the outer level is a feedback control loop that periodically adjusts the scheduling weights and server allocations of the inner level.

A profile-driven performance model for cluster-based multi-component online services is presented in [129]. Offline constructed application profiles characterize component resource needs and inter-component communications. The performance model differentiates remote invocations from fast-path calls between co-located components and they measure the network delay caused by blocking inter-component communications. This work also explores how the performance model can be used to assist system management functions such as optimized placement, capacity planning, and cost-effectiveness analysis.
Comparison

Our work differs from the efforts in the areas of high performance computing, web server systems, and multimedia applications primarily in terms of infrastructure, applications, resource management desired explained below.

The computing environment we consider is of a utility system hosted in data centers. The data centers consist of virtualized, dedicated, shared, heterogeneous resource elements. We are also investigating the support of QoS guarantees to applications, in addition to achieving high system utilization through, for example, load balancing alone. We consider a different set of applications, those of desktop applications hosted within remote desktop sessions, than the applications - batch, telnet, web requests, or multimedia such as streaming video. In terms of supported resource management functions, unlike the existing works for web servers and multimedia applications where admission is controlled for a web session request or a multimedia session request into a web server or multimedia server, we additionally control the admission of applications such as web server and multimedia server themselves into a utility system. We also support a greater diversity of applications than web or multimedia server alone. We share with QualMan the approach of developing the resource management infrastructure as a middleware that can be deployed on existing commodity systems.

Our work differs from the efforts in the areas of Multi-Tier systems and Web Services in terms of the application domain considered for demonstration of the model-driven resource management techniques. We consider remote desktop sessions which present additional challenges due to the user-driven workloads and the remote display technology. We also consider multi-capacity resource assignment where multiple resources are taken into consideration.

8.2.2 Classification By Approach

In here, we classify related work based on the approach used for resource management. We present the work in categories of queuing-models, feedback control systems, profiling and
trace-based techniques.

**Queuing-Models Based**

Numerous projects have used queuing theory. Here, we present recent works, with application to resource management.

An analytical model based on a network of queues is presented in [76] for multi-tier internet services. The queues represent different tiers of the application. The model is sufficiently general to capture (i) the behavior of tiers with significantly different performance characteristics and (ii) application idiosyncrasies such as session-based workloads, tier replication, load imbalances across replicas, and caching at intermediate tiers. This work also demonstrates the utility of the model for dynamic capacity provisioning, performance prediction, bottleneck identification, and session policing.

A model-based approach to utility resource management focusing on coordinated provisioning of memory and storage resources is presented in [77]. The approach uses models derived from basic queuing theory. The models are used to predict the value of candidate resource allotments under changing load. The performance measures predicted by Web models are the CPU response time, Object cache hit ratio, Storage request load in IOPS, Average storage response time, Average total response time.

In the work on performance modeling for multi-component online services presented in [129], the average response time at each server is determined using an M/G/1 queue. This response time prediction is used to assist system management functions such as component placement, capacity planning, and cost-effectiveness analysis.

**Feedback Control Systems**

Several existing works have explored the use of feedback control systems in the design of resource management systems. We present a few of them below.

The work on optimizing Apache [130] explores online optimization techniques of Apache
Web Server, by optimizing the settings of configuration parameters. This work shows the effect of MaxClient configuration parameter on the response time. They explore two optimizers to find the optimal value of MaxClients. Another related work explores the optimization of Database Servers [131]. It further handles interdependencies between configuration parameters. The optimization technique used is Nelder-Mead simplex method.

A QoS architecture for a shared storage proxy cache which can provide service differentiation to competing classes has been described in [132]. They propose the use of per-class feedback controllers that control the space allocation for a given class so as to track a specified target hit rate.

Other works are [70, 68, 69].

Profiling and Trace-Based Techniques

We summarize below few efforts in this area, and provide references to others.

An analysis of CPU utilization traces from a shared utility computing environment is presented in [133]. In this work, PCA (Principal Component Analysis) technique is used to characterize each applications’ workload. The applications are characterized by three features - periodic, noisy, and spiky.

Profiling has been used in [65] to derive an accurate estimate of application resource needs. The derivation is in aid of demonstrating the feasibility and benefits of overbooking resources in shared hosting platforms to maximize the platform yield. The applications profiled are Apache web server, MPEG streaming media server, Quake game server, PostgreSQL database server.

GRACE-OS is described in [134]. It is an energy-efficient soft real-time CPU scheduler for mobile devices that primarily run multimedia applications. It uses online profiling and estimation to obtain the CPU cycle demand distribution of the multimedia applications. Such a demand distribution is then used to make scheduling decisions. The approach of using a demand distribution derived using profiling provides statistical performance guarantees to
the applications.

Other related work in this area are those of [135, 129, 136, 137, 138]

8.2.3 Comparison

In summary, in our research, we consider a combination of statistical and analytical (queuing) techniques for deriving performance models. We also consider dependencies and combine the statistical and analytical model relations. These relationships are derived for desktop applications hosted in a utility environment.

We primarily focus on the application of models for initial resource allocation functions such as admission control and resource assignment of desktop sessions. We do not address runtime control of the desktop sessions in this work.

8.3 Service Oriented Architecture

Service-oriented architectures (SOA) are a continuation of traditional component-based development [139]. Service-oriented computing enables trends toward autonomy and heterogeneity at the global scale. SOA represents a tie between various areas, such as Grid computing, autonomic computing, and enterprise computing, by enabling underlying mechanisms for implementing policies and controls for these different domains.

We have based our work on the emerging trend of service oriented architectures.
Chapter 9

Conclusions and Future Work

In this chapter, we present our final conclusions and propose areas for future work. We begin by revisiting our hypothesis and showing the benefit of model-driven approach. Thereafter, we summarize our contributions and lessons learned. Finally, we present future work and open issues.

9.1 Revisiting the Hypothesis

We stated in the Introduction chapter our hypothesis that

*Application model-driven resource management in shared utility systems will yield better combined application QoS guarantees and system efficiency, than over-provisioning (worst case) or under-provisioning (best-effort) of resources.*

In order to validate the hypothesis, we conducted an experiment for the e-mail application. The experimental setup for the e-mail application is similar to that described for the modeling derivation in Chapter 5. Specifically, we use KMail as the e-mail client and VNC as the remote display technology. We apply the send workload to the e-mail client. The scenario is to host three e-mail client sessions in a remote desktop utility. The SLA desired for each session is: 25 second or less for the screen update response time, and 6-7 minutes or less for the service time. We use each of the three approaches – overprovisioning, underprovisioning (best-effort), and model-driven to achieve the same.

With an overprovisioning approach, we assign a single server for each of the e-mail client sessions. With an underprovisioning (best effort) approach, we assign a single server for all
the three sessions. With a model-driven approach, we assign the allocations according to the model. This is visually illustrated in Figure 9.1. For the desired SLA of 25 second or less for screen update response, the allocation value provided by the model is 40 percent. As this value also satisfies the requirement for service time, we decided to choose this as the CPU share to be allocated for our experiment. With a 40 percent CPU allocation, we would need 120 percent allocation for hosting the three e-mail client sessions. This is achieved by hosting two sessions on one machine, and one on another machine.

Figures 9.2(a) and 9.2(b) illustrate the CPU consumptions with overprovisioning and best-effort respectively. The average CPU utilization with an overprovisioning and best-effort approach is 0.15 and 0.51 respectively. With best effort it is 0.51. With the model-driven approach, the resource partitioning software, HP PRM [80], was used to guarantee 40 percent CPU share for each client session. Figure 9.2(c) illustrates the CPU consumption for the server with two sessions. The average CPU utilization is 0.29.

Figure 9.3 shows the cumulative probability distribution of screen update response time for the three approaches. Table 9.2 specifically compares the 95th and 50th percentile values
Figure 9.2: CPU Utilization with (a) Overprovisioning, (b) Underprovisioning (best-effort) (three sessions execute simultaneously), (c) CPU Shares decided using Models (each session gets a 40 percent CPU share allocation)

for the three approaches. As shown, with best effort, the response time is quite high and above the desired SLA of 25 seconds or less. Even the 50th percentile value is higher than 25 seconds. On the other hand, with overprovisioning, one achieves a very good screen update response time. The 95th percentile value is 10.15 second much below the desired SLA of 25 seconds of less. Thus, with overprovisioning, the user has a very good experience and the QoS requirements are well satisfied. However, this has come at the expense of system efficiency. As pointed out earlier, with overprovisioning, the average CPU utilization is only 0.15. With the model-driven approach, the 95th percentile value is 25.68 seconds as had been predicted using the model equations. Thus, with the model-driven approach, one not only satisfies the SLA desired, but also achieves a better system efficiency than overprovisioning.

Table 9.3 compares the service times for overprovisioning, best-effort, and model-driven approaches. With overprovisioning, the service time is 355 seconds which is below the desired
SLA of 6-7 minutes or less. With best effort, the service time is 522 seconds which is greater than the desired SLA. With model-driven approach, we achieve the predicted 415 seconds satisfying the SLA. Similar arguments as used for the screen update response time holds true for service time as well.

Figure 9.4 maps these results into the hypothesis illustration introduced in Chapter 1. As can be seen, compared to overprovisioning and underprovisioning (best-effort), a model-driven approach provides better combined application QoS guarantees and effective system efficiency.

Before moving to the next section, we would like to point out a note. One observes from Figure 9.2 that that the workload can be decomposed cleanly into three parts. The CPU consumption during the middle half of the applied workload is relatively much smaller.
Figure 9.4: Illustration of the Benefit of Model-driven Resource Management

compared to the earlier and later parts. If one would decompose the workload in this manner and modeled each part separately, the overall system utilization using a model-driven approach could be further improved.

9.2 Contributions and Lessons Learned

We now summarize our major contributions and lessons learned.

9.2.1 Summary of Key Contributions

Methodology for Model-driven Resource Management

We define a general application model consisting of a QoS-, dependency-, workload-, and performance-model. We describe a methodology to derive such an application model. In summary, the proposed methodology is to (i) characterize workloads and dependencies, (ii) profile application usage in dedicated environment, (iii) forecast application behavior using statistical models on profiled data, (iv) complement and extend profiled relationships with analytical models, (v) extend the relationships in presence of application dependencies, (vi) validate the derived models. Once the application models are developed, they are applied towards resource management functions in a utility system.
Deriving of Application Models in Remote Desktops

We demonstrate the general methodology of model-driven resource management for a remote desktop utility. We have created novel models for remote desktop sessions. These models provide prediction of resource usage for a desktop session. The modeling derivation for the remote desktop models is split into two stages. In the first stage, the derivation process is applied to each application to be executed in the desktop session. We illustrate this derivation by example of an e-mail application. The considered application QoS parameters are those of service time for e-mail send/receive operations and screen update response time. The workload is characterized as a mix of send and receive e-mail operations with a message size distribution. The profiling is done using a send workload, and the relation of the service times and screen update response times to CPU share allocation is determined through statistical models. An analytical model is created using queuing theory to model the response time from an e-mail server.

In the second stage, we apply a timing dependency structure for a set of applications that execute in the desktop session. Timing dependencies refer to the execution order in which applications are started within a session. The derivation consists of two steps. First, we extend the performance model relations from the first stage, applying the dependency structure, and obtain the performance model equations for the remote desktop. Second, we design mechanisms for dependency characterization. We propose analysis of system usage logs for statistical dependency characterization. A dependency matrix maintains the probability of sequential and simultaneous execution of applications, which is then used at run-time to infer the value of the dependency structure for a given set of applications.

Design and Validation of Resource Management for Desktop Sessions

We have designed resource management modules that are driven by the remote desktop models. Specifically, we have developed a Site Admission Control and a Resource Assignment system that rely on the remote desktop models to obtain the predicted resource allocation
shares. The resource assignment problem is formulated as an on-line, multi-capacity, bin-packing problem. Multiple resource types correspond to the multiple capacities of bins. We define a weighted fitting metric that represents the fit between the resources desired by the model, and the resources actually available in the system. We then use a Best Fit algorithm to fit as many requests as possible in the system.

We have built a prototype implementation to validate the feasibility of the design. The prototype represents a remote desktop utility system. Users submit requests for desktop sessions to the utility. The predicted system resource shares required to meet user SLA requirements are determined from the model equations. These predicted values are fed into admission control module, that makes admission decision for the desktop session. If the session is admitted, the resource assignment module assigns a share of resources specified by the model. We use Globus software [22], which we have extended to further support interactive sessions [23].

We have also built a simulation analysis tool that implements the proposed resource management solution. Using the tool, we performed simulation studies to demonstrate benefits gained through a model-driven approach. Specifically, we emulate a real-world scenario by running experiments for a mixed workload consisting of batch and remote desktop sessions [24]. Using a model-driven approach, we show the advantage of sharing a single pool of resources for both types of jobs (batch and desktop). This leads to cost saving of resources, while at the same guaranteeing application QoS guarantees, without much degradation in throughput and wait time.

9.2.2 Lessons Learned

We now enumerate the lessons learned from our major contributions.

- Model-driven design is useful for automation in next generation data centers to deal with increasing complexity and to improve performance.
• **However, generalization of the modeling results is difficult.** What is applicable is the methodology.

• **Remote desktop sessions provide challenges in deriving models.** Challenges arise due to user driven workloads, multiple applications, remote display technology, remote storage.

• **Application Services are continuously changing.** Any modeling mechanism has to keep up with the trend of application complexity. This can be an issue in a model-driven design, wherein frequent changes in the model could trigger corresponding design changes.

• **SLA guarantees provided are only as accurate as the models.** Accurate model generation is hence a key requirement.

• **It is hard to automate the derivation of models in dynamic environments.** In most cases, the derivation is not completely accurate, rather it relies on heuristics, and it is statistical. However, once the model is derived, it becomes relatively easy to automate management functions. Hence, creating and building the models is the most difficult part in model-driven resource management.

### 9.2.3 Use of Results

We consider it critical to have useful application of our contributions. We summarize below some potential uses of our contributions.

• Applying the methodology presented in this thesis for performance management in utility systems can lead to development of useful tools. Specifically, the steps of identifying QoS model, categorizing dependency structure, and identifying workload and user model, can be used to derive benchmarking and workload generation tools. Similarly, the step of profiling application instances in a dedicated environment can lead
to development of measurement tools to collect data for the metrics identified in the QoS model.

- The derived remote desktop session modeling equations can be used to derive rules of thumb regarding the design and configuration of the utility system. For example, the derived rules could drive sizing tools that estimate the resources to purchase for satisfying performance requirements of popular workloads. The model equations can also be developed into a set of configurable tools, that could be executed with different design configurations and for different workload requests.

- The designed resource management systems, admission control and resource assignment can be developed as a set of tools for dynamic resource management. Further, the simulation tool and obtained results can be used for design evaluation, and the results obtained using it can be used to derive best practices in the presence of scale, application mix, and heterogeneity.

9.3 Future Work and Open Issues

Some of the areas for future work of the presented contributions are:

- Modeling for multiple resources. In this thesis, we demonstrated single application modeling with respect to CPU allocation only. However, the performance of modern day applications is also affected by allocations made for other resources, e.g. memory and network. Modeling the effect of memory on application performance, both independently and in correlation with CPU is an interesting topic. Furthermore, the effects of latency in deployments where the clients are geographically distributed cannot be ignored. Modeling the effect on an end-user perceived performance metric such as screen update response time with different latency/bandwidth conditions is a promising topic for future work.
• *Modeling the effect of virtual machines.* Virtual machines bring about overhead that affect the overall performance. An interesting topic is to model the I/O overhead in a remote desktop utility wherein the I/O (display and storage) is remote.

• *Modeling using traces from real usage.* Simulated workloads enable the modeler to control the workload parameters and for deriving the performance relationships. However, the approach would not be scalable to model several hundred applications. Simulated workloads also do not aid in inferring time of day or weekly trends in resource usage. Modeling using traces of actual system usage by users would help overcome these limitations.

• *Run-time derivation and refinement of models.* In this thesis, we have illustrated the use of models for initial resource allocation. At runtime, dynamic workload variations may cause the prediction made by the models to become inaccurate. One would need to research mechanisms to do run-time derivation and refinement of models. Even if the workload is stable, the data collected at runtime can be used to further refine the models. Exploration of these mechanisms is a promising area for future research.

We now conclude the thesis with a few broader open issues and questions.

• *Ownership of application models.* The introduction of application models for performance management raises an interesting issue as to who should be responsible for creating such models. Should it be the application provider? Should it be the infrastructure provider? Or should it be third-party vendors who would infer these models using historical and black-box techniques?

• *Automating the creation of performance models.* Clearly, the derivation of models is time consuming. For statistical models, the modeler needs to analyze the data, and then repeatedly change the configurations to derive new modeling equations that would satisfy the requirements. Analytical models require careful analysis of the application
so as to understand the queuing effects, the arrival pattern, the service time etc. What could we do to reduce the model derivation time? Could we build heuristic driven tools to automate the choice of configuration parameters for statistical models? Could we have automated tools that would analyze the queuing effects in complex distributed applications? Could we extrapolate models to infer predictions for different sets of configurations from predictions made for a certain configuration?

- **Deriving techniques that is in between a pure white box and a pure black box approach.**

There are two extremes in model derivation approaches. One of them is a pure black-box based approach that relies only on the system level trace data to derive model predictions. It does not require application knowledge. However, this approach faces limitations in use cases such as a remote desktop utility where workloads are user driven. The other extreme is a pure white box approach, wherein one requires a detailed knowledge of the application structure and behavior for performance predictions. This approach works well when the set of popular applications is small. However, it may not always be feasible, and it cannot scale when there are large number of applications. Can we derive mechanisms that takes an approach in between the two extremes, perhaps a hybrid approach? Can we leverage best practices provided by experienced system administrators managing the application in meaningful ways?
References


143


[31] University of Illinois. Plato.


Vita

Vanish Talwar was born in April, 1978 in India. He received his Bachelor of Technology degree in Computer Science from Institute of Technology, Banaras Hindu University (IT-BHU) in May 1999. He then joined the Computer Science Department in the University of Illinois at Urbana Champaign for graduate studies in August 1999. In Summer 2000, he worked on the TSpaces project as a summer intern at IBM Almaden Research Center in San Jose, California. He received his Master of Science degree in Computer Science in May 2001. Subsequently, he joined Hewlett-Packard’s Embedded Software Operation. In August 2002, he moved over to Hewlett-Packard Research Labs in Palo Alto, California. He continued to make progress towards his PhD studies under the supervision of Prof. Klara Nahrstedt while also working at HP Labs.

His general technical interests include distributed systems, operating systems, and computer networks, with a focus on management technologies in utility systems. He is the recipient of the David J. Kuck Best Masters Thesis Award in the Department of Computer Science at UIUC and the Jawaharlal Nehru Summer Research Fellowship. He has been elected to the Honor Society of Phi Kappa Phi and to Sigma Xi, The Scientific Research Society. He has served on the program committee for MGC’05 - Workshop on Middleware for Grid Computing co-located with ACM/IFIP/USENIX Middleware 2005.