PERFORMANCE AND CONTROL OF PARALLEL MULTI-SERVER QUEUES
WITH APPLICATIONS TO WEB HOSTING SERVICES

A Thesis in
Industrial Engineering
by
Yiyu Chen

© 2006 Yiyu Chen

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2006
The thesis of Yiyu Chen was reviewed and approved* by the following:

Soundar R.T. Kumara  
Distinguished Professor of Industrial Engineering  
Chair of Committee, Co-advisor

Natarajan Gautam  
Associate Professor of Industrial Engineering  
Co-Advisor

Anand Sivasubramaniam  
Professor of Computer Science and Engineering

Dennis K.J. Lin  
Distinguished Professor of Supply Chain and Information Systems

Richard J. Koubek  
Professor of Industrial Engineering  
Head of the Department of Industrial Engineering and Manufacturing Engineering

*Signatures are on file in the Graduate School.
This thesis addresses power and performance management of a queueing system as follows. It consists of multiple parallel queues, each of which serves single type of requests and is called an application. For each queue, there are possibly multiple servers, with some scheduling discipline and waiting area having infinite capacity. Each request brings a random amount of work and will be assigned by application scheduler to a server and processed at a particular speed. Web hosting center is an example of such a system. Therein operates thousands of servers consuming huge amount of energy and serves hundreds of applications that require the system to deliver Quality of Service (QoS). The complexity of managing such huge systems in cost effective manners is not a trivial endeavor, especially under stringent QoS requirements from self-similar like traffic.

This thesis proposes a three-tiered solution methodology at system, application, and server level as a tunable resource allocation strategy for power and performance management (for such aforementioned systems). At system level, a centralized $G/G/m$-based solving algorithm, applicable at different time-granularities, is developed to provision servers at required operating frequencies. A packing/scheduling mechanism is built for dispatching incoming workload into a given set of servers to balance energy-savings and performance needs in the second tier. Further, in the third tier, a decentralized frequency control strategy for an individual server is devised by adopting fluid modeling and Markov Decision Process (MDP).

In terms of contributions, this thesis provides an energy aware resource management framework for web hosting centers. The proposed three-tiered solution methodology can achieve significantly high amount of energy-savings and maintain the system performances as well, compared to those done by prior studies. Secondly, it is one of the first few studies, which provides a rigorous optimization formulation under self-similar-like workloads with respect to availability of controllable resources. Solutions derived from those optimization can be easily computed and implemented in an online fashion for computer systems. Finally, the proposed solution methodology can be generalized as a strategy of resource management for any parallel multi-server queueing system serving multiple classes of requests, such as call centers.
# TABLE OF CONTENTS

List of Figures vii

List of Tables ix

Chapter 1 Introduction 1
  1.1 Resource management issues ........................................ 2
  1.2 Performance modeling and controlling mechanisms issues ....... 3
  1.3 Main contribution .................................................. 5
  1.4 Thesis organization ................................................ 6

Chapter 2 Problem Statement 8
  2.1 System specification of web hosting centers .................... 9
  2.2 Generalized queueing system .................................... 12
  2.3 Resource control for power-performance management .......... 15
  2.4 Proposed methodologies ......................................... 17

Chapter 3 Related Studies 20
  3.1 Performance-Power management .................................. 20
  3.2 Characterization and prediction for workloads ................. 21
  3.3 Performance measures ........................................... 23
    3.3.1 Time-varying and long range dependent arrivals .......... 23
    3.3.2 Heavy-tail distributed workload .......................... 24
  3.4 Control mechanisms .............................................. 25

Chapter 4 System Level Optimization 29
  4.1 System model ..................................................... 31
  4.2 Problem formulation ............................................. 33
  4.3 Solution Methodologies ......................................... 36
    4.3.1 Prediction for time varying traffic parameters .......... 37
      4.3.1.1 Arrival process ....................................... 38
      4.3.1.2 File size ............................................. 41
    4.3.2 Control mechanisms ......................................... 42
4.3.2.1 Queuing analysis ................................................. 42
4.3.2.2 Algorithm for the non-linear optimization .................. 43
4.3.2.3 Control theoretic approach and hybrid approach .......... 46
4.4 Experiments .............................................................. 47
  4.4.1 Experimental set up .............................................. 47
  4.4.2 Results .............................................................. 51
    4.4.2.1 Need for sophisticated strategies ......................... 51
    4.4.2.2 Comparison with IBM method ............................... 53
    4.4.2.3 Comparing the proposed schemes ......................... 54
  4.4.3 Varying the parameters ........................................ 57
4.5 Concluding remarks .................................................. 59

Chapter 5 Application Level Optimization 61
  5.1 System model ....................................................... 64
    5.1.1 Servers ....................................................... 64
    5.1.2 Services/Clients ............................................ 65
    5.1.3 Client workloads: a case for co-location .................. 66
    5.1.4 Metrics and overheads ...................................... 68
    5.2 Problem formulation ............................................ 69
  5.3 Solution methodologies ............................................ 75
    5.3.1 Co-location/packing of client workloads .................. 75
    5.3.2 Algorithm for proactive DVS ................................. 79
  5.4 Experimental Evaluation and Results ............................. 80
    5.4.1 Experimental setup ........................................... 80
    5.4.2 Results ........................................................ 82
      5.4.2.1 Comparing packing mechanisms ............................ 83
      5.4.2.2 Comparing DVS mechanisms ............................... 86
  5.5 Concluding remarks ................................................ 89

Chapter 6 Server Level Optimization 91
  6.1 System model and problem formulation .......................... 93
  6.2 Solution methodologies ........................................... 96
    6.2.1 Fluid approximations for server workloads ................. 99
      6.2.1.1 CTMC .................................................... 100
      6.2.1.2 DTMC .................................................... 103
    6.2.2 MDP formulation ............................................. 104
    6.2.3 Structure of optimal policy .................................. 107
  6.3 Experiments .......................................................... 109
    6.3.1 Experimental set up ......................................... 109
    6.3.2 Compare with DTMC and CTMC ............................... 110
      6.3.2.1 Approximation of system transition ..................... 110
      6.3.2.2 Varying the parameters ................................ 112
        6.3.2.2.1 Effect of window size ............................... 114
        6.3.2.2.2 Traffic intensity .................................. 115
    6.3.3 Compare with existing DVS heuristics ..................... 116
  6.4 Conclusion Remark .................................................. 118

Chapter 7 Conclusion 120
Chapter A System Level Admission Control

A.1 Background and System Overview .................................................. 126
A.2 Client Perceived Pageview Model .................................................. 130
  A.2.1 Average Pageview Response Time .............................................. 133
  A.2.2 Server-end Response Time ....................................................... 136
  A.2.3 Failure Rate ........................................................................... 138
A.3 Control Mechanism ....................................................................... 139
  A.3.1 Steady-State Control Optimization ............................................. 140
  A.3.2 Transient SYN Management ..................................................... 143
    A.3.2.1 Observed Inter-arrival Time ............................................... 143
A.4 Experiments Evaluation ................................................................. 144
  A.4.1 Service Requirement Inference .................................................. 146
  A.4.2 Validation of Decomposed Pageview Model ............................... 147
  A.4.3 Transient Control ................................................................. 149
    A.4.3.1 Interarrival Time Control ................................................. 149
A.5 Conclusion .................................................................................... 150

Bibliography .................................................................................... 152
LIST OF FIGURES

2.1 System Model ................................................................. 10
2.2 System Schematics .......................................................... 13
2.3 Scheme of Tasks .............................................................. 13
2.4 Performance and Control Windows ......................................... 16
2.5 Schematics of Proposed Three-tier Structure ............................. 17

4.1 System Schematics at System Level ........................................ 32
4.2 Granularity ($T, t$) for Server Allocation and Frequency Modulation ........ 36
4.3 Collecting Window-wise Statistics Scheme .................................... 39
4.4 Time-varying Arrival Rates .................................................. 48
4.5 Service-time Requirements for Files of Different Sizes ....................... 50
4.6 Time-varying Behavior for Appln. 2 and Appln. 3 in WL3 .................... 55

5.1 System Schematics at Application Level .................................... 63
5.2 Client Resource Usage Traces ............................................... 66
5.3 CDF of CPU Demand with $\eta$ Capacity Assurance ......................... 71
5.4 Pseudo-code for Packing ..................................................... 76
5.5 Pseudo-code for DVS Algorithm ................................................. 80
5.6 Spill-overs for Hour 47 with Optimal DVS .................................. 85
5.7 DVS Profile of Time and Energy .............................................. 88

6.1 System Schematic at Server Level .......................................... 94
6.2 Schematics of Fluid Model .................................................... 99
6.3 Dynamics of Fluid at the Buffer ............................................. 100
6.4 Transitions of States from CTMC and DTMC Modeling ....................... 111
6.5 System Performance with Fixed Frequency Control ......................... 113
6.6 DTMC v.s. CTMC at $\tau = 5, r = 1$ ........................................ 113
6.7 DTMC v.s. CTMC at $\tau = 15, \gamma = 1$ ...................................... 114
6.8 DTMC v.s. CTMC at $\tau = 5, \gamma = 0.8$ ..................................... 115
6.9 DTMC v.s. CTMC at $\tau = 15, \gamma = 0.8$ .................................... 116

A.1 Example of pageview service .............................................. 127
A.2 Example of failure of second connection .................................... 128
A.3 SYN retransmission ........................................................... 129
A.4 Example of a pageview RT on two connections .............................. 132
A.5 Transition based on scenario two connections ............................ 135
A.6 Server Models ...................................................................... 137
A.7 Inter-arrival Time Control Algorithm ................................. 144
A.8 Experimental testbed. ......................................................... 144
A.9 Open Loop ........................................................................ 146
A.10 Close Loop ........................................................................ 147
A.11 Validation of decompose pageview. ....................................... 148
A.12 Response Surface for IAT control ........................................ 150
### List of Tables

3.1 Comparision with related work ........................................... 26
4.1 For a 256 KB file, service-times at different operating frequencies .... 49
4.2 Default parameter values for simulation ..................................... 51
4.3 Results for the “simple” strategies ........................................ 53
4.4 Comparison between schemes on savings of electricity-cost ............... 54
4.5 Results for workload with 30 applications and 285 servers ................. 57
4.6 Results for $B_0 = 0$, WL3 .................................................. 58
5.1 Notations and definitions used in this chapter .............................. 69
5.2 Simulation parameters ...................................................... 80
5.3 Naive Packing for client sessions ............................................ 81
5.4 Comparing the schemes for Packing Algorithm and DVS Schemes. ........ 83
6.1 Notations and definitions in chapter 6 ...................................... 96
6.2 Parameter and values for Synthetic Traces ................................ 110
6.3 Comparisons of DVS with $\tau = 4$ minute ................................ 117
6.4 Comparisons of DVS with $\tau = 1$ minute ................................ 117
A.1 Notation and Explanations .................................................. 131
Introduction

The motivation for this work stems from two growing trends of web hosting systems serving customers on a commercial basis. One trend is that the requirement of huge amount of resources and effort in management of the system is leading numerous enterprises to offload their applications/tasks to web hosting centers. Web hosting centers cater to the computation needs for these applications with a very large pool of resources and earn revenues. Examples of such applications are finance transactions, scientific computing, and e-commerce transactions; and the various resources are thousands of servers, data storage devices, etc. Because of the inherent differences in nature of the large number of applications, the resource management cost and overall complexity for the hosting centers increase greatly.

The other growing trend is that the customers require a guaranteed level of Quality of Service (QoS) instead of a best-effort service (a prevalent trend during earlier days). QoS is typically specified in terms of system performance such as response time, availability, accuracy, etc. These customers can be regarded as individual applications, where QoS levels are decided by Service
Level Agreement (SLA). Hence, web hosting centers need to allocate and manage resources according to this SLA. Nevertheless, this task becomes even more complex and more difficult in the presence of a workload which no longer follows the well-studied Poisson process, but shows self-similarity (long range dependence and heavy-tailed distribution). Obtaining performance measures with a self-similar workload is not straight-forward and is difficult to implement. These trends lead to interesting research issues, such as large scale resource allocation and optimization of system revenue, while simultaneously maintaining SLA under self-similar traffic at various degrees. This thesis investigates and addresses these issues.

1.1 Resource management issues

The goal of resource management for web hosting centers is to deploy resources in a cost-effective manner to increase profit. A web hosting center collects revenues by serving the applications that are charged according to the provided services (decided by SLA). The current SLAs strongly require the QoS values to be met for each application. Failing results in the loss of revenues or the incurrence of penalties. Common performance measures used in SLA are response times, information loss-rate, etc. An application requests performance bounds (such as specific target response-time) on these measures during its execution. The applications require different SLAs in terms of measures and QoS values. This requirement of a separate treatment for each application leads to differentiated services, which adds more complexity in maintaining system performance.

Web hosting centers maintain and deploy enormous numbers of servers, storage devices, networks supplies, mechanical equipment, and cooling systems to provide guaranteed system performances to all users’ requests. The energy consumption by such a system is becoming a serious concern. The electrical costs of operating servers system often end up being far greater than the initial hardware deployment cost. To quote Eric Schmidt, (C.E.O. of Google), “What matters most to computer designers at Google, is not speed, instead of power - low power; because data
centers can consume as much electricity as a city.” It points out that a bottleneck of system design lies in power supplies. One also needs to be concerned with the environmental issues when generating and delivering such a high amount of electric power. Over the past few years, issues of operating those servers and deploying system resources in an economically effective way have attracted researchers’ efforts on proposing various resource allocation and control mechanisms.

The challenges of resource management in web hosting centers comes from the workloads across the applications characterized by (i) time-varying behavior, i.e. arrival rate changing with time and (ii) self-similar property, i.e., long range dependence (LRD) and heavy-tailed (HT) distribution of traffic parameters. Due to the time-varying workloads, static and conservative resource allocation leads to excessive energy consumption; on the other hand, static and stringent resource allocation causes performance degradation. Self-similarity of web traffic requires more resource allocation to meet the requirements of SLA, compared to well-studied queueing system with independent and light-tailed distributed workloads. Therefore, without any modification, performance measures derived from standard queuing theory are not well suited to serve as the basis for controlling and allocating resources of web hosting system. The difficulty of devising energy-aware resource allocation mechanism lies in the balance of ever stringent power constraint of providing resource and higher resources demand of serving workload with self-similarity property.

1.2 Performance modeling and controlling mechanisms issues

Appropriate and economical resource allocation mechanism depends on the well-suited performance modeling of critical resources and workloads. The workload of applications at web hosting centers display time variability in an unsynchronized manner. In addition, inter-arrival times of web workloads vary a lot and show varying degrees of correlation at different time scales. Full
knowledge of workload transitions can leverage the limited resources among applications that experience different workload. The prediction mechanism of critical traffic parameters, therefore, needs to accommodate time variability and long range dependence with respect to prediction horizon. So far, only complicated characterization models [45, 88] and simple fixed prediction models [45] have been proposed to address prediction issues of time-varying traffic parameters. There is a necessity to develop a refined and easily-implementable prediction mechanism for significant traffic parameters for obtaining performance measurement and achieving efficient resource allocation.

System performance under self-similar traffic has a higher chance to experience from system overload for a longer time because the heavy-tailed distribution of workload takes a longer time to reach a steady state. Often, standard queueing models, such as $M/M/m$ (which cannot predict performance of web hosting centers) have been illustrated by researchers [98, 35], who have also emphasized the need for more appropriate models. A lot of self-similar performance models have been proposed in the past few years. Fluid model have been used as approximation for self-similar workload in a steady state and transient case. Because of the fast-varying nature of workloads, transient analysis is more demanding than steady state analysis when it come to designing a refined control mechanism.

Resource and performance control mechanism aiming to meet the whole spectrum of SLAs may be a costly option or may not even be a plausible solution due to the limited power supply and highly varying workload. A practical and implementable scheme would be to balance the trade-off between decreasing resource allocation for reducing energy consumption and to increase resource provisions for meeting SLA under self-similar traffic. Therefore, performance control and resource allocation of web hosting centers is driven by workloads. Web hosting traffic appears different statistical characteristics at application level and request under different time scale. Server provisions and mapping influence applications’ performances at course time granularity. On the other hand, the scheduling for assigning servers and operating speed of servers have a
significant impact on refine granularity control.

1.3 Main contribution

The research undertaken in this study is unique in several ways, being the first endeavor to provide a multi-tiered solution (namely, at system, application and request levels) for resource allocation and energy optimization for a multi-server queueing system at web hosting centers. Indeed, it melds the disjoint areas of stochastic processes, operating systems, and optimization to develop multi-tiered performance and control schemes. The specific contributions of this study are as follows:

- It provides a energy saving solution for multi-server queueing systems such as web hosting centers. It not only saves on power consumption of such a system by a significant amount, but also maintain required system performances. It also demonstrates that minimization of energy and benefits of performance are inter-dependent and are tunable depending upon the higher-level policies or specific objectives. The nature of the solution is generic in nature; nevertheless, is applicable to scenarios with specific goals. The flexibility of the solution allows adoption of any combination of tiers (of proposed schemes) to cater to different needs of different customers. The solution is robust as it degenerates to the default behavior in worst case.

- It develops a novel and easily-implementable prediction mechanism for time-varying and long range dependent traffic parameters that can be applied on standard queuing analysis and performance measurement of self-similar traffic.

- It designs a service control mechanism that deploys two knobs for control decisions, such as provisioning servers and adjusting frequency, for all applications to minimize energy consumption of the whole system and also satisfy applications' SLA. The knobs are sepa-
rately usable, should the situation require so. Thus, the mechanism is portable enough to facilitate diverse requirements.

- It develops a centralized frequency controller for applications in coordination with server allocation and server mapping for energy minimization and performance sustainance.

- It provides an energy-aware services/clients packing mechanism for each application based on requests’ workload statistics, such as mean and variance of mean file size. It not only saves energy but also provides fair performances to the clients, regardless of class.

- It applies new performance metrics to investigate the trade-off between energy and performance with respect of two common power management mechanisms, server allocation(on/off) and frequency control. Reliability is one critical factor considered in this trade-off, reducing the overall management and procurement cost.

- It develops a decentralized proactive heuristic for frequency control based on the state of the load and prediction error of the load at server level. This facilitates agility in responsiveness at each of the server on one hand and increases scalability of prediction to the workload changed.

- It provides a decentralized optimal frequency control mechanism based on the fluid model considering the nature of heavy tailed distribution of the workloads. The structure of optimal policy is to increase the frequency of a server corresponding to reminding server workload and can be easily implemented in an on-line fashion with negligible computational overhead.

1.4 Thesis organization

This thesis first provides system specification of web hosting centers under consideration and the problem statement in chapter 2. The objectives of each tier of web hosting center, such as system
level, application level and server level, are defined with the necessary details. Three crucial tasks of 1) workload prediction, 2) performance measures, and 3) optimization of resource control are required for the problem solving and methodology development at these levels. The framework of multi-tiered performance-energy control is proposed to handle all three tasks. Related studies of those tasks defined are summarized in chapter 3. The reviews of control mechanisms involved in the system, application and server level are also detailed in the last section of chapter 3.

The thesis elaborates the motivation, system specification, detailed problem statement, and solution methodology of each level separately in chapters 4, 5, and 6. In chapter 4, the thesis considers the whole system of multi-applications and provides a framework of workload prediction mechanism, server allocation mechanism across all applications, and centralized frequency control mechanism at the application level. For single application, the thesis develops a request-to-server mapping algorithm to improve system performance and energy consumption by balancing highly-varying workloads in chapter 5. The proposed packing algorithms are evaluated with a developed heuristics of decentralized frequency control. As a third-tier solution, this thesis finally proposes an optimal decentralized frequency control mechanism based on fluid modeling to tackle down heavy-tailed workloads in chapter 6. Relative pros and cons for the proposed solution methodologies conclude this thesis, with a small discussion on future directions in chapter 7.
This study addresses issues of energy-aware resources management and performance control under time-varying, long range dependent, and heavy-tail distributed workloads for multi-server queueing systems. The motivation of this thesis, web hosting center, is one of the systems that have urgent needs in energy-performance management and face such type of workloads. Therefore, this study first provides the system specification of web hosting centers, with extra focus on the perspectives of power consumption and management in the first section. Following that, this study illustrates a generalized multi-server queueing system, its performance requirements, and characteristics of workloads in section 2.1. It further defines the corresponding tasks to perform on-line minimization of power consumption and fulfillment of performance requirement for such a queueing system. Finally, in the last section, this thesis proposes a multi-tiered solution to integrate all the defined tasks with respect to different levels of system abstraction.
2.1 System specification of web hosting centers

The focus of this study is a web hosting center containing a number of identical servers, called a server cluster, which are all equally capable of running any application given to them. Applications ranging in a diverse spectrum (from scientific domain to the commercial world) service client requests. Each request is directed to one of the servers running the application, which processes the request in a time (service time) that is related to a request parameter (e.g. file size). Consequently, one can impact the throughput/response time for these requests by modulating the number of servers allocated to an application at any time.

Dense blade systems are being increasingly deployed in hosting centers because of their smaller form factors (making it easier to rack them up) and power consumption characteristics. Such systems provide a fairly powerful (server-class) processor such as an Intel Xeon, 1-2 GB of memory, and perhaps a small SCSI disk. If there are more intensive I/O demands, then a NAS/SAN is used to interconnect these blades to a more extensive storage device/array. There can be several hundreds/thousands of these blade servers in a hosting center, all consuming electric power depending on whether they are turned on, and if so at what frequency. Investigation in this study is targeted at the power management of these blade systems/servers.

From the viewpoint of the customers of the hosting center, it is important to have at least as many servers as necessary for that application in order to meet a desired level of performance (service level agreement - SLA) for its client requests. This study aims to bound the average response time for requests as SLA, and a more extensive SLA could have been used as well. From the viewpoint of the hosting center, the goal is to consume as little electrical energy as possible while still ensuring that the individual applications are able to meet their end-user SLAs for client requests. A schematic of the system environment is given in Figure 2.1.

The workload imposed on a server’s resources when running an application expends electric

\footnote{The term, \textit{customer}, is denoted as an application that is being hosted by the center, and each application in turn may need to serve requests (e.g. HTTP requests) from different clients on the Internet.}
power, thereby incurring a cost in its operation. The important resources of concern include semiconductor components such as the server CPU, caches, DRAMs, and system interconnects, as well as electro-mechanical components such as disks. There are two direct mechanisms available today for managing the power consumption of these systems:

- One can temporarily power down the blade, which ensures that no electricity flows to any component of this server. While this can provide the most power savings, the downside is that this blade is not available to serve any requests. Bringing up the machine to serve requests would incur additional costs, in terms of (i) time and energy expended to boot up the machine during which requests can still not be served, and (ii) increased wear-and-tear of different components (the disks in particular) which can reduce the mean-time between failures (MTBF) leading to additional costs for replacements and personnel.

- Another common option for power management these days is dynamic voltage/frequency scaling (DVS). As will be shown in section 4.2, the dynamic power consumed in integrated circuits is proportional to the cubic power of the operating clock frequency. Slowing down
the clock allows the scaling down of the supply voltages for the circuits, resulting in power savings. Even though not all server components may be exporting software interfaces to perform DVS, CPUs [6], even those in the server market, are starting to allow such dynamic control. With the CPUs consuming the bulk of the power in a blade server (note that an Intel Xeon consumes between 75-100 Watts at full speed, while the other blade components including the disk can add about 15-30 Watts), DVS control in our environment can provide substantial power savings. The overheads of transitioning between frequencies for DVS are negligible (requiring just a few clock cycles). However, a more extensive algorithm for frequency setting may be needed in order to provide a good trade-off between performance and energy savings.

DVS implementation for a server cluster of these blades can be broadly classified based on whether (i) the control is completely decentralized, where each node independently makes its scaling choice purely on local information, or (ii) there is a coordinated (perhaps centralized) control mechanism that regulates the operation of each node. Though decentralized DVS control is attractive from the implementation viewpoint, previous research [33] has shown that a coordinated voltage scaling approach can provide substantially higher savings. The intuitive underlying reason is that a system with two nodes operating at the same frequency $x$ consumes lower power than a system with one node running at 1.5$x$ and another at 0.5$x$, assuming the workloads are divisible among the servers.

The proposed framework allows the employment of both these options towards enhancing energy savings, without compromising on end-user SLAs. In the proposed model, when a machine is switched off, it consumes no power. When the machine is on, it can operate at a number of discrete frequencies $f_1 < f_2 < ... < f_\ell$, where the relationship between the power consumption and these operating frequencies is of the form $P = P_{\text{fixed}} + P_f f^3$ (see section 4.2), so that it captures the cubic relationship with the CPU frequency while still accounting for the power consumption of other components which do not scale with the frequency.
On the other hand, those mechanisms are also available for SLA management by providing sufficient service capacity. The scheduling and packing of requests into the allocated servers within an application can also have a huge impact on requests’ perceived system performance, with given service capacity. Size-based scheduling has been widely applied to improve the fairness and response time of the web hosting system [73]. Smart scheduling and packing clients into servers therefore can be considered as a third mechanism to indirectly save power consumption by achieving better SLA.

2.2 Generalized queueing system

The system described in subsection 2.1 can be modeled as a queueing system with standard operation research terminologies. The queueing system considered in this thesis is schematized in figure 2.2. It consists of a pool of identical servers and multiple application queues, each of which represents an individual application. When a request, requiring certain file size, arrives to the system, it will enter an application queue according to its identity and wait in the buffer to be processed. Each application requires guaranteed QoS, which is measured in terms of response time, queue length, packet loss etc. Note that an application considered here can be thought of as a class that has different performance requirement. Requests, waiting in each application queue, will be dispatched by a given scheduler to a set of servers assigned to that application specifically. Service time of a request depends on the size of the requested file and the server speed. Allotted servers of different applications can be operated at different speeds, depending on the workload for each application. Thus, at a specific instant of time, each application can be viewed as an independent system of a single queue and multiple servers.

The arrivals and service time experienced by such a queueing system has the following three characteristics: i) time variability, ii) LRD arrivals, and iii) HT service time distribution. The requests from different applications have different periodic arrival patterns that are affected by
file size is highly varying and results in a heavy tail probability density distribution. Requests show a high correlation that is the phenomenon of LRD arrivals. Moreover, requested choices with traffic parameters approximated as constants, the inter-arrival times of subsequent requests might be approximately measured as piece-wise constant parameters. Given a set of servers and speed choices, the traffic parameters, such as mean arrival-rate and mean file-size, can be approximately measured as piece-wise constant parameters. Given a set of servers and speed choices with traffic parameters approximated as constants, the inter-arrival times of subsequent requests show a high correlation that is the phenomenon of LRD arrivals. Moreover, requested file size is highly varying and results in a heavy tail probability density distribution.

Figure 2.2. System Schematics

Figure 2.3. Scheme of Tasks
In order to maintain system performances under aforementioned traffic and achieve the main objective of this thesis, power saving, control of service capacity and power consumption of such a system need to be coordinated well. Service capacity of the system is controlled by each node of server, which can be turned off or on at different speeds and dedicated to different individual applications. Power consumption of such a system is composed of nodes of operating servers and its operating speed, usually ranging in 8 discrete levels \( f \). The common rule of thumb for this relationship is given by \( V \propto f \). Since the power consumption of all other components (except the CPU) is independent of the frequency, the following simple model of power consumed by one server node running at frequency \( f \) is assumed by this thesis: \( P = P_{\text{fixed}} + P_f \cdot f^3 \). The energy consumption of the servers allocated certain application \( i \) at the hosting center operating at a constant frequency \( f \), over time \( t \), as

\[
E_i = M_i \cdot \int_t P \cdot dt = M_i \cdot (P_{\text{fixed}} + P_f \cdot f^3) \cdot t,
\]

where \( M_i \) denotes number of server allotted to application \( i \). Due to the time-varying and heavy-tailed distributed workloads, \( M_i \), and server frequency, \( f \), need to be controlled over time to manage the energy consumption while adhering to any performance goals. At the same time, the scheduler of each application can dispatch the requests according to the corresponding server allocation and their workloads statistics. Frequency of each allotted server can be operated depending on the workloads sent by scheduler. If a server experiences a highly varying workload, it has a higher chance or running at a higher frequency and results in a much higher power consumption due to the cubic relationship of frequency between energy and frequency.
2.3 Resource control for power-performance management

This thesis considers the following control variables: 1) server on-off, 2) server speed, 3) server-application-request mapping. It is well understood that server on-off operating can takes several seconds/minutes, consuming extra energy in process. On the contrary, the server frequency adjusting incurs negligible machine time. Therefore, this thesis assumes that operations related to server on-off, such as sever-application-request mapping, can be executed in a larger time frame; frequency adjusting of server is assumed to have no operating cost and be excused in any time frame. In addition to the constraint of execution of control variables, the design of control mechanisms of such a web hosting center also needs to accommodate workloads transition that dominates the system performance. Here, workload statistics can be obtained only through online prediction during the course of designing and implementation. Performance measures of SLAs under time-varying and long range dependent arrivals and heavy-tailed workloads also show disparate degree of variation at different time scales. Selection of the time window is critical for obtaining system performance measures and conducting energy-aware controlling.

Various optimizations with respect to different modeling time are, therefore, required to bridge energy consumption, stochastic performance models and control mechanisms. Due to the limitation of resources and the high requirement of resources from heavy-tailed workloads, optimization mechanism has to balance between the energy cost and revenue lost from performance violation to certain requests at different level of system abstraction. For example, during the large window, the optimization for the whole the system may focus on balancing energy consumption across applications while just adhering to their average requirement of SLAs. The optimizer at application level may utilize more detailed workload statistics to achieve a better balance of low energy and high system performance by a smart scheduler. In summary, the tasks required for solving the aforesaid problem are following: 1) prediction mechanism of traffic parameters, 2) performance measure of time-varying, LRD, and heavy-tailed workloads, and 3) optimization of
control mechanism as depicted in figure 2.3.

In order to obtain performance metrics and control decisions, this study intends to break up a huge time horizon into several non-overlapping time windows. Within each time window, the traffic parameters are approximated and are held constant. The traffic parameters under the time-varying pattern can be well approximated as piece-wise constant. The performance measures described in the second tasks also need be deployed for time interval where the traffic parameters remain fairly constant and long enough for the system to reach a steady state. Moreover, applying control mechanisms at a fixed window granularity is a cost-effective strategy. Therefore, the three tasks considered in this thesis are approached in a window-based measurement fashion, where a window means time interval of a specific length (as shown in figure 2.4). In the figure 2.4, the time-horizon has been divided into $n$ windows. Before the $n^{th}$ window starts, the system is supposed to predict system performance based on traffic parameters and then execute controlling actions to ensure the QoS for $n^{th}$ window for any application. Note the determining proper window size is a subtask in this thesis and will be discussed in section 3.

Note that the above queuing system is a generic case for the system that has multiple servers, multiple queues, and multiple classes. The performance analysis and controlling mechanisms derived in this study are applicable to wide spectrums of systems with such components such as web hosting centers, call centers, etc.
2.4 Proposed methodologies

This study proposes a multi-tiered performance and resource control mechanism based on availability of resource at each tier to manage ever increasing complex performance-energy issues for web hosting centers. As depicted in figure 2.1 in section 2.1, three resource control knobs are considered in this thesis: 1) server allocation to applications, 2) scheduling policy to map applications’ client into their allotted servers and 3) dynamic voltage scaling of each server. The cost and time associating with first and second control variables are higher than the third, even with today’s advance in technology. Hence, it is expected to design and execute the first and second control actions in a coarse granularity and DVS can be controlled in a much refined granularity, especially for the decentralized scheme.

Depending on the availability of resources and control variables at different time granularities, this study defines three levels of control methodology at the system level, application level and server level, each of which requires tasks (i) traffic parameter prediction, (ii) performance measures, and (iii) design of control mechanism.

1. System-level solution methodology: the system-level solution (the first tier) focuses on
allocating overall resources (especially servers) across all applications with unsynchronized
time-varying workloads and performance requirement. The optimization span is usually in
a unit of a week, where time variability of applications often follows predictable patterns.
It focuses on the average load and performance at coarse granularity. Therefore, this study
considers only two control knobs, server allocation and centralized DVS, at this level. The
scheduler is assumed to fixed as First Come First Serve (FCFS) policy, that expects to
achieve the balanced average load on each server. All related details are illustrated in
chapter 4.

2. Application-level solution methodology: the application-level methodology (the second
tier), intends to utilize wider spectrum information of clients’ requests to design a more
sophisticated scheduler packing requests into allotted servers in coordination with server
allocations. Schedulers at each application aim to pack/co-locate types of services from
clients into a subset of a given number of back-end servers that is bounded by the server
provision from system-level solution. Since the decision variable is restricted by the system
level, the original multi-period packing problem at the scheduler of each application can
be viewed as a single period (such as one hour) packing problem. The packing/co-locating
algorithm takes advantages of the correlation among clients’ requests so that clients with
different service requests still feel fair about the service because of similar perceived de-
lay on each server. Moreover, due to the short span of optimization and indivisibility of
client workload, this study proposes a feedback-driven decentralized DVS mechanism that
investigates the benefit of power saving from all available control knobs in the stochastic
environment. All related details are illustrated in chapter 5.

3. Server-level solution methodology: dynamic frequency control of a single server can be
executed in finer granularity, either coordinating with other servers or not. Centralized
frequency control is expected to achieve lower energy by utilizing more workloads informa-
tion and supports of balanced server loads (through smart packing). Therefore, in reality, decentralized frequency control at single server can even outperform centralized frequency control in terms of providing stringent system and individual users-perceived performance.

This study proposes the third-tier performance control methodology, optimal decentralized frequency algorithm, that is based on fluid modeling to tackle the nature of heavy-tail distributed workloads. The optimal algorithm only utilizes the state of the system at control point and can be combined and executed with proposed system-level and application-level solution methodology. All related details are illustrated in chapter 6.

Note that the methodologies proposed in this study can be applied integrally or separately. In order not to be limited by the scenarios considered in this study, this study will also provide the motivation, implication and application of each tier individually in the following chapters to expand the scope of this study.
This chapter discusses all the related work about performance and energy control of parallel-server queues for web hosting centers. First, an overview of resource-power management at web hosting center is given. The following sections study the related work of three main tasks defined in 2.3. It focuses on the related studies on traffic prediction mechanisms of web traffic parameters in the second section. Various of performance models of self-similar traffic are investigated in the following section. The last section gives an overview of control mechanisms for performance-power issues of web hosting centers at the system, application, and server level (as described in section 2.4).

### 3.1 Performance-Power management

Since many of the services/applications hosted at these centers can have stringent service-level agreements to be met, there have been several investigations into the resource capacity planning and dynamic provisioning issues for QoS control (e.g. [12, 69, 84, 101]). As many studies have pointed out, over-provisioning of resources can be economically unattractive, and it is
important to accommodate transient overload situations (which are quite common for these services) with the existing resources [23]. Dynamic load monitoring (e.g. [67]), transient-load based optimization (e.g. [23, 95]) and feedback-based techniques (e.g. [10, 38, 51]) are being examined to handle these situations. However, these studies have mainly focused on performance (and revenue based on an SLA) and have not examined the power consumption issues.

Earlier, optimizing power consumption has been of primary importance in mobile devices to extend battery life by shutting down inactive components or with Dynamic Voltage Scaling (DVS) [24, 96]. It is only recently that power management has gained prominence in the high-end server market. The high operating frequencies mandate very expensive cooling solutions, from the chip-level to the datacenter and machine-room level [63]. Further, the cost of electricity generation and delivery is a cause for concern in operating large datacenters. Consequently, evolving microprocessors for the server space have also started supporting multiple power states and DVS capabilities as in the Intel Irwindale.

3.2 Characterization and prediction for workloads

The workload of web hosting centers has been known for its self-similar characteristics and drawn research interests on such traffic, especially on characterization and performance measures. M. F. Arlitt and C. L. Williamson [13] presented the first work systematically characterizing arrival process and file size distribution of web traces. They found out that arrivals of web traffic are time dependent, and long range dependent and file size distribution is heavy-tailed. Sahinoglu and Tekinary [77] showed both data sources as well as real time applications generate traffic patterns that exhibit certain degrees of correlation between arrivals, and show long-range dependence. Several experiments and measures have shown that other traffic types, such as WWW traffic (Crovella and Bestavros [30]) and Wide Area Network traffic (Lucas et al. [52]), all exhibit LRD as well.
Squillante et al. [88] modeled the number of arrivals within a stationary window using the ARMA(p,q) model. The dependence structure of the arrival process highly affected the queue length and workload distribution. The higher the correlation of arrivals, the worse the system performance. The fixed model has been widely applied to model the periodic arrival process [45]. Squillante et al. [45] proposed using the phase type distribution for parameters in ARMA to characterize the transition among the piece-wise time series. The Matrix geometric technique is applied to obtain the transition point of coefficient of variation in each stationary series. It involves a complex computation routine and characterizes the dependent and piece-wise stationary process and perform comparably well to Flintstone methodology with SURGE methods [15]. Time-varying and long range dependent traffic parameters of web hosting center have been well characterized. However, the prediction mechanism of arrivals, that is critical of on-line mechanism, has not been addressed earner.

To the author’s best knowledge, related studies [13, 16] have been devoted to characterize the file size distribution; however, there is no research that has focused on the issues of prediction of file size. In most of cases, file size is assumed to be known for controlling system [8]; or the average file for the the whole trace is used for modeling purpose. The average values of file size from applications are not good indicators for file size of future arrivals, because the distribution of file size of the web server is commonly characterized as heavy-tailed distribution. It either has high value of variance or even infinite variance. Consequently, it is difficult to characterize and predict the exact distribution of file size. [36], Riska et al. [73] proposed to approximate heavy tailed workloads by mixture of exponential distributions. Both of the methods provide resealable approximation of file size distribution but require intensive off-line computation that is not preferable in a real time control environment at web hosting center.
3.3 Performance measures

Performance measure, such as average response time, throughput and queue length of self-similar traffic have been developed from two perspectives: 1) time-varying and long range dependent arrivals and 2) heavy-tail distributed workloads. This thesis summarizes the related studies in those two categories in the following sections.

3.3.1 Time-varying and long range dependent arrivals

Models addressing the time-varying/long range dependence to analyze self-similar traffic can be classified into the following different categories:

1. Model time-varying traffic. Massey et al. [53, 54] and A. Ross [74] developed performance analysis on time-varying queues as a non-homogeneous Poisson process, assuming the traffic parameters at time $t$, such as arrival rate $\lambda(t)$, are assumed to be known.

2. Model arrival time using auto-correlated stochastic process with given parameters (known as Hurst parameters). Chan and Li [22] define $\{X_n|n \geq 1\}$, a stochastic process characterizing the inter-arrival times. For asymptotic second order self-similar traffic of a single queue with deterministic service times, Tsybakov and Georganas [94, 93, 92] obtain performance bounds including the queue length process for heavy tailed workloads.

3. Model arrival streams using an on-off or more complex fluid source to take care of the heavy correlation. Boxma and Dumas’s survey paper [19] sheds some light into this model. Typically the on-time is heavy-tailed. Jelenkovic and Lazar’s [46] develop an approximate expression for the tail distribution of the workload when the source is on-off and on-time heavy tailed.

4. Model a generalized traffic resource by extending the ideas of Markov modulated Poisson process. In all these cases an environment process governs the traffic generation. The
sojourn times is typically heavy tailed. Schwefel and Lipsky [80] consider N on/off sources (not fluid, but discrete) with peak and mean burst given, with heavy-tailed burst. Queue length and delay distribution has been investigated. This technique is similar to model 2 above with the exception that discrete arrival bursts are used instead of fluids.

5. Model the self-similar process using renewal process with inter-renewal times according to a heavy-tailed distribution. Greiner et al. [41] modify the GI/M/1 results where the arrival process is renewal with heavy-tailed distribution. They obtain distributions of queue length and waiting times.

As described in Gautam and Seshadri [39], the second method is theoretically sound, but due to the mathematical intractability it is not favored much. Method 5, in particular, ignores the correlation between the inter-arrival times and therefore is not a very suitable model. The fourth method provides a solid ground for deriving delay from the perspective of burst; however, the methods of obtaining important parameters, such as on/off time and burst size, are not provided there. Hence, it is difficult to implement the fourth method on the web hosting center. Due to the high volume of traffic from applications floating into the web hosting centers, the performance measures based on fluid modeling serve a good candidate for resource and performance control.

3.3.2 Heavy-tail distributed workload

The performance analysis of heavy-tailed distribution has focused on the tail distribution of response time [97, 36, 11, 73]. There are two families of heavy-tailed distribution: the Pareto distribution and the Weibull distribution [36]. The Weibull distribution (with Probability Density Function (PDF) \( P(x) = \alpha \beta^{-\alpha} x^{\alpha-1} e^{-(x/\beta)^\alpha} \)) has a finite mean and finite large variance. On the contrary, the Pareto distribution (with PDF \( P(x) = \frac{\alpha x^\alpha}{\beta^{\alpha+1}} \)) has a possibly finite mean if \( \alpha > 1 \), but possibly infinite variance when \( \alpha < 2 \).

Previous research considered finite and infinite variance. For a single server case, Whitt [9]
derived the exact tail distribution through the exponential mixture of inverse Gaussian distributions and proved the analysis by Cohen [29] as a special case in their studies. Whitt [36] also proposed to use a mixture of exponential distribution to approximate heavy-tailed distribution during a finite interval to avoid the lack of Laplace-Stieltjes-Transforms (LST) for the distributions with infinite variance. Based on Whitt [36], Riska et al. [73] proposed a computation algorithm to fast compute the tail distribution of response time in $M/PH/1$ system. For the multiple servers cases, Whitt [97] derived the lower bound of tail distribution of response time in $M/G_{HT}/m$ and showed the convergence rate from transient state to the steady state. Following on Whitt’s framework [97], Wolf [11] assert that $m$ slower servers are always better than one fast server in terms of tail distribution of response time in the regime of heavy-tailed file distribution. He also provided necessary and sufficient conditions, which are defined by higher moments, a function of $m$ servers and traffic intensity $\rho$, for a delay moment in a FIFO multi-servers queues.

### 3.4 Control mechanisms

Due to the complexity of resource management and performance measures, design of control mechanism for web hosting centers to achieve a win-win solution of performance-power has been addressed in system, application, and server level, depending availability of resources and workloads. The whole system of web hosting center aims to allocate servers among allocation in a cost-effective manner for the long run. For each application, an independent subsystem, control mechanism has to coordinate with decision of allotted servers to dispatch requests among them. Meanwhile, all individual servers can be operated in different speeds either in consistent with timing of servers-allocations mapping or not. Earlier studies have been centered on system and server level, which are summarized in section 3.4 and 3.4. The related studies at the application level are provided in section 3.4.
**System Level** It is only recently that energy management for server clusters found in hosting centers has gained much attention [21, 26, 34, 43, 49, 66]. Amongst these early studies, [25, 26, 66] show that resource allocation and energy management can be intertwined by developing techniques for shutting down servers that are not in use or have a low load (by offloading their duties to other servers). A detailed study of the power profile of a real system by researchers at IBM Austin [17] points out that the CPU is the largest consuming component for typical web server configuration. Subsequent studies [33, 82] have looked at optimizing this power by monitoring evolving load and performance metrics to dynamically modulate CPU frequencies. A categorization of these most closely related investigations is summarized in Table 3.1, in terms of whether (i) the schemes consider just server shutdowns or allow DVS, (ii) they focus on energy management of just 1 server (which can be thought of as completely independent management of a server without regard to the overall workload across the system), (iii) they consider different applications to be concurrently executing across these servers with different loads, (iv) they try to meet SLAs imposed by these applications, and (v) they incorporate the cost of rebooting servers (in terms of both time and MTBF).

<table>
<thead>
<tr>
<th></th>
<th>DVS?</th>
<th>Multiple servers?</th>
<th>Multiple applns.?</th>
<th>SLA?</th>
<th>Reboot cost?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duke [25]</td>
<td>No</td>
<td>No</td>
<td>Yes(5)</td>
<td>Yes</td>
<td>Yes(Time)</td>
</tr>
<tr>
<td>Rutgers [66]</td>
<td>No</td>
<td>Yes(8)</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Virginia [82]</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IBM [33]</td>
<td>Yes</td>
<td>Yes(10)</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Table 3.1.** Comparison with related work

Of the four related studies shown here, only two [33, 82] have considered the possibility of DVS for server applications. Of these two, only the IBM work [33] has examined the issues in the context of a server cluster, with the other focusing on a single machine/server setting. However, even the IBM work has not considered the issues in the context of multiple applications being hosted on these server clusters (i.e. the server provisioning problem in conjunction with energy management), nor have they considered the long term impact of machine reboots on operating
Application Level  The requests of an application are mapped and dispatched into the allotted server according to their type, which could be defined by the requested task, size, or random selection. Servers, dedicated to that application, are responsible to one to many types of requests, depending on the size of workloads. The concept of merging and mapping same types of requests into multi-servers system is the same as Server-Based/Thin-Client Computing that is gaining popularity of advantages. As shown in table 3.1, capacity planning and dynamic resource provisioning of single application has been studied extensively, e.g. [23, 7, 101]. However, they are either not in the content of client workload consolidation nor consider the multiple-servers and SLA. Moreover, a number of commercial products target to consolidate client workloads at the back-end servers [62, 72, 83, 1, 2, 7, 70]. The performance issues with such consolidation, especially for interactive performance, have also been studied in detail [79, 56, 100, 14, 99]. However, there is little prior work on resource provisions simultaneously considering co-location and power management, when consolidating client workloads. To our knowledge, this is the first paper to study the issue of how these client workloads need to be co-located on the allotted servers to achieve such consolidation, from both performance and power perspectives.

Server Level  Dynamic voltage scaling is one of the most popular power management control tool for the single processer. Control mechanism of DVS is mainly studied for single CPU in a decentralized manner, since energy management has traditionally been considered important for mobile and resource-constrained environments that are limited by battery capacities. The influence that the frequency (which allows the voltage to be scaled down) has on the power consumption has been exploited to implement DVS mechanisms for energy management of integrated circuits [24], [75]. Consequently, many processors - not only those for the mobile market, but even those in the server space [6] - today are starting to offer software interfaces for DVS. There have been numerous studies on exploiting DVS for power savings without compromising
on performance (e.g. [37], [42], [50], [64], [?]). The aforementioned studies have been for mobile
devices that have very different workload patterns than servers, and/or for embedded environ-
ments where soft/hard real-time constraints need to be met within a power budget. [82] and
[33] have developed control heuristics of DVS in a decentralized manner and centralized manner
separately for processors at web hosting cluster. However, the characteristics of workloads have
not been fully utilized to obtain the optimal frequency/voltage setting for web hosting servers.
System Level Optimization

The growth of network-based commercial services, together with the off-loading of IT services, is leading to the growth of hosting/data centers that may need to house several applications/services. From the perspective of data centers, it has to cater to the disparate needs of several applications, making the task much more difficult because of the size of the problem. Given adequate amount of resources, over-provisioning the service capacity at such centers to allow for worst case load conditions is economically not very attractive. Consequently, much of prior work (e.g. [23, 69, 95]) has looked at finding the right capacity, and distributing this capacity between the different applications based on their SLAs. Similar argument holds for energy, which is being looked as another form of resource. As several recent studies have pointed out [25, 26, 47, 49, 66] data centers can consume several Megawatt of power. It has been observed that future centers could add 5GW of demands (which is around 10% of the current generating capacity for California) [26], putting the requirements at approximately 40 TWH (costing around $4 billion) per year. It is not just the cost of powering these servers, but we need to also include the cost of designing and deploying cooling systems (which in turn consume power) to ensure
stable operation. We are at power densities of around 100 Watts per square feet, and the cooling problem is expected to get worse with shrinking form factors. Finally, one also needs to be concerned with the environmental issues when generating and delivering such high electric power capacities. Statically provisioning the number of servers for a given cost (say of electricity) and performance SLA can be conservative or miss out on opportunities for savings, since workloads are typically varying over time. Dynamic power management is thus turning out to be very important when deploying server farms/clusters, and hardware has started providing assistance to achieve this. Much of the earlier work [25, 66] in this area, used server turn off/on mechanisms for power management. The only other work which considered a combination of these two mechanisms (turn off and DVS) was done at IBM [33]. However, this work did not (nor did the others) consider the impact of server off/on cycles on the long term reliability of server components due to wear-and-tear. Further, none of the prior studies have really considered the goal of meeting a response time requirement (SLA). Instead, they have tried to optimize energy first, at a slight degradation in response time.

As results (in section 4.4) will show, not treating the performance-based SLA in the dynamic optimization as a first class citizen causes these other schemes to fall short, even though they may provide more energy savings. Finally, most of the prior work has just looked at optimizing the energy in a single application setting, assuming the existence of a higher level server provisioning mechanism across applications. Framework presented in the following sections, integrates server provisioning across applications (which automatically decides how many servers to turn on/off) with energy management within an application, in an elegant fashion while still allowing different options for these two steps. This chapter presents a dynamic energy optimization framework for server clusters for all hosted applications at hosting centers and makes the following contributions: (a) It presents the first formalization of this dynamic optimization problem of server provisioning and DVS control for multiple applications, which includes a response-time SLA and the costs (both power and wear-and-tear) of server shutdowns. Even though in the proposed
solutions the optimization is decoupled into two steps - server provisioning and DVS control, these two are still closely intertwined. (b) Two new online mechanisms are proposed, and one other feedback mechanism is implemented to solve this problem. The first is pro-active in that it predicts workload behavior for the near future and uses this information in a stochastic queuing model to perform control. This study implements feedback control in [27] as reactive benchmark mechanism to achieve the same goals. The third is a hybrid scheme in which this study uses predictive information from the first for server provisioning, and feedback control of the latter for DVS.

This chapter proposes a methodology to the optimization of resource allocations, performance maintainable, and energy saving in regard to the whole web hosting center. First, the actual system is abstracted into the trackable model in section ?? . The detailed problem statement is formulated in section 4.2. Proposed methodology and experiments are presented in the following sections 4.3 and 4.4 respectively.

4.1 System model

This chapter considers simple scheduling policy of "First Come and First Serve" (FCFS) to dispatch the request to available servers within applications. The more sophisticated scheduler will be investigated in chapter 5 to improve system performance. This study considers a coordinated DVS strategy and expects to bring in more power saving than decentralized DVS under the premise of balancing load that is achievable in average under FCFS. A controller is assigned for each application, and it periodically assigns the operating frequency/voltage for the servers running that application. This chapter performs this at an application granularity since the load for each application and corresponding SLA requirements, can be quite different. The structure of this two- level controller is schematized in figure 4.1. Further, in this model, each server continues to be entirely devoted to a single application until the time of server reallocation. Since
both server re-allocation and time granularity for DVS are fairly coarse (around 20-60 minutes for re-allocation and 5-30 minutes for DVS), the overheads of centralized control (which would probably add just a few seconds at most over distributed control) are quite negligible.

With this model as in figure 4.1 of the hosting center, the two solution steps to the problem at hand are to (i) perform server provisioning to decide how many servers to allocate to each application, and (ii) decide what should be the operating frequency for the servers allocated to each application. These steps need to be done periodically to account for time-varying behavior for the arrival and provisioning requirements. Note that the first step may require bringing up/down servers and/or moving servers from one application to another. The time cost for such migrations of applications ($T_{migrate}$) and/or reboots ($T_{reboot}$) are incorporated in model
considered in the following sections. Further, there is a long term impact of server reboots due to wear-and-tear of components such as the disk and the dollar cost associated with such factors is included.

4.2 Problem formulation

This chapter models hosting center as a set of parallel stations in a queueing system with $M$ servers which are all identical and are equally capable of running any application. This model allows a maximum of $N$ different applications to be hosted at any time across these servers, with each application $i$ being allocated $m_i$ servers at any time ($\sum_{i=1}^{N} m_i \leq M$). When a server is operational, it can run between a maximum frequency $f_{max}$ (consuming the highest power) and a minimum frequency $f_{min}$, with a range of discrete operating frequency levels in-between. The service time of a request that comes to an application is impacted (linearly) by the frequency, and the overall service capacity allocated to an application is also directly related to the number of servers allotted to it.

As shown in equation 2.1, the energy consumption of a hosting center is decided by the number of on-servers and their operating frequency over time. Consequently, there are two ways of controlling the overall energy consumed by all the servers at the hosting center: (i) powering/shutting down the servers that are not really needed (i.e. reducing $M$), and (ii) scaling the operating frequency ($f$), and a combination of one or both of these may be used depending on the mechanisms exported by the underlying system and the overall costs. Since the real workload is expected to be time varying, this thesis may need to control the number of active servers and their frequency dynamically over time based on the evolving workload. Let’s say that these are controlled periodically at a granularity of time $t$. 
Electricity Cost of Operation: Over a duration of $Z$ such time units of duration $t$, the overall energy consumption of our hosting center can be given as

$$
\sum_{z=1}^{Z} \sum_{i=1}^{N} m_i(z) \ast (P_{fixed} + P_f \ast f_i(z)^3) \ast t,
$$

where $m_i(z)$ is the number of servers allocated to application $i$ during the $z$-th time period which are all running at a frequency $f_i(z)$. If $K_S$ is the electricity charge expressed as dollars per unit of energy consumption (e.g. kilowatt hour), then the total electricity cost for operation of the hosting center is

$$
\sum_{z=1}^{Z} \sum_{i=1}^{N} K_S \ast m_i(z) \ast (P_{fixed} + P_f \ast f_i(z)^3) \ast t,
$$

expressed in dollars.

Cost of Server Turn-ons: One important point which these discussions ignore (and which is also ignored in much of the earlier work [33, 66, 25]) is the impact of turning on/off servers. While frequency scaling is itself relatively simple (achievable within a few cycles on most current hardware), and may not significantly affect the long term reliability of the system, the effects of turning off/on the servers should not be overlooked. Machine reboots can last several seconds/minutes, consuming power in the process. Further, it is well understood that server components (disks in particular) can be susceptible to machine start-stop cycles [31], thus reducing the Mean-Time-Between-Failure (MTBF) when we perform server shut-downs during the execution for energy savings. If $B_o$ is the dollar cost of a single server turn-on cycle, then the total amount of dollars expended by any mechanism using this approach can be calculated as

$$
\sum_{z=1}^{Z} B_o \ast \left[ \sum_{i=1}^{N} m_i(z) \ast \sum_{i=1}^{N} m_i(z - 1) \right] ^+.
$$
Note that the term $x^+ = x$ when $x \geq 0$ and $x^+ = 0$ otherwise. $B_o$ itself needs to account for the dollar energy cost for bringing up the machine ($P \times T_{reboot} \times K_S$), together with the dollar cost incurred by a smaller MTBF (denoted as $C_r$), i.e. $B_o = P_{max} \times T_{reboot} \times K_S + C_r$, where $P_{max}$ denotes the server power consumption when running at $f_{max}$.

Putting together the above two factors, we get the dollar cost of operation to be

$$Z = \sum_{z=1}^{N} \left( \sum_{i=1}^{N} K_S \times m_i(z) \times (P_{fixed} + P_f \times f_i(z)^3) \times t + B_o \times \left[ \sum_{i=1}^{N} m_i(z) - \sum_{i=1}^{N} m_i(z - 1) \right]^+ \right)$$

which is the objective function to minimize. Note that one could choose to minimize this objective by inordinately slowing down the frequencies, or shutting down a large number of servers (just keeping 1 active to serve requests). This can result in violating any service level agreements (to meet a response time requirement) that the hosting center may have agreed to with the customer providing the application. In this exercise, this chapter uses a simple SLA (a more extensive one could be used as well), where the requirement is to meet an average response time target (that is set by a prior SLA). Consequently, when minimizing the above objective function, one need to obey the following constraints:

- $W_i \leq \bar{W}_i$, which is the SLA to constrain the average response time of application $i$ to a target $\bar{W}_i$.
- $\sum_{i=1}^{N} m_i(z) \leq M$, where $m_i(z) \in Int$. Since the total number of servers allocated to applications has to be less than the hosting center capacity.

Note that, this thesis only considers $f_i(z) \in F = \{f_1, f_2, ..., f_\ell\}$ where $f_1 < f_2 < \ldots < f_\ell$, which says that a running server has to operate at one of the $\ell$ discrete frequencies. $f_{min}$ is used to denote $f_1$ and $f_{max}$ to denote $f_\ell$. 

4.3 Solution Methodologies

The whole system under consideration is modeled as a set of $N$ parallel queues denoting the $N$ applications as elaborated in schematic Figure 2.2 in chapter 2. Client requests for application $i$ arrive one by one into its corresponding infinite capacity queue (queue-$i$). To accommodate time-varying workload behavior, this study assumes that every $T$ minutes, servers are allocated for the $N$ applications (i.e. for all $i$, $m_i$ is determined). Likewise, every $t$ minutes, the frequency $f_i$ for the $m_i$ servers of application $i$ is determined. Although not required for modeling, for logical and implementation reasons, this study assumes that $T$ is an integer multiple of $t$. Thereby at larger time granularities $T$, server allocation/provisioning is performed, and at smaller time granularities ($t$), the servers are tuned to different frequencies. The basic premise for doing this at 2 granularities, with $T > t$ is that server re-allocation (machine reboots and application migration) is much more time-consuming than changing the frequency. There are $U$ intervals of length $T$ present in the entire duration for which this study wants to optimize, and each of these (denoted by $u = 1$ to $U$) has $S$ intervals of length $t$ (i.e. $s = 1$ to $S$). This is pictorially shown in figure 4.2.

Setting $m_i$ and $f_i$ by analyzing this queuing system involves three phases. First, request arrival pattern and service time requirements need to be predicted, as described in subsection 4.3.1. As mentioned earlier, the proposed workload prediction approach can be applied to a wide range of web traces, independent of performance measures and control mechanisms. Next,
the predicted arrival and service time information will be used in a queuing model to determine
the mean response time for each application in the following subsection. Finally, in the last
subsection, the response time information is utilized in the proposed optimization problem, for
which this study describes a tractable solution strategy. Therefore, this study names the proposed
approach as **SQNO** (steady-state queueing based nonlinear optimization approach).

### 4.3.1 Prediction for time varying traffic parameters

To perform queuing analysis in each interval, it requires estimates of mean arrival rate of requests,
variance of arrival rate, mean file size in bytes, and variance file size in bytes for each application.
Notice that all four of the above parameters are time varying as well as stochastic, requiring a
prediction of their values at each interval. Historically, researchers have used either self-similar
traffic patterns or used deterministic time-varying parameters. On the one hand, self-similar
models are attractive for describing the nature of the traffic and generating synthetic workloads,
but they do not lend themselves well to analysis. On the other hand, time-varying models require
Poisson assumptions and full knowledge of the time-varying behavior.

This section intends to characterize and predict non-stationary and periodic web traffic pa-
rameters by a simple technique that can easily predict future traffic parameters. The periodic
pattern of arriving requests of the applications lightens up the viability of building up prediction
models for the traffic parameters. However, the traffic also shows a considerable amount of
development from the fixed periodical behavior; therefore, the complexity of building up a sound
predicting model increases. The assumption of independence of input data, such as inter-arrival
times and file size over a window, is critical for applying queuing analysis on a stationary traffic.
Hence, it is critical not only to obtain predicted values of traffic parameters during a stationary
window but also to have the independent input for predicting traffic parameters. **It is impor-
tant to note that it is not our goal here to determine the best prediction technique.** However,
our contribution is that if the parameters are well predicted, the inter-arrival times and file size
requirements can be approximated as independent and identically distributed (i.i.d.) inside each interval and provide a viable and reliable prediction base for conducting on-line queuing performance analysis. The next few subsections first talk about the prediction analysis of arrival process and then file size separately.

4.3.1.1 Arrival process

For the arrival process, this thesis predicts two most important arrival parameters in queuing analysis: mean arrival rate and variance of arrival rate. A SARMA (Seasonal Autoregressive Moving Average) /ARMA (Autoregressive Moving Average) is proposed to model and predict the mean and variance of arrival rate across windows in the next few paragraphs. The selection rule of the prediction window size is also provided in the following paragraph.

Model  Suppose the sequence of time-varying request observations from a Web server environment of interest is such that each observation is indexed by its time parameter $t$. This series of requests is partitioned into $M$ windows, each of which is an input for time-series process, $Y_m$, where $Y_m$ is a random variable corresponding to the mean or variance of arrival rate of the $m^{th}$ time-series window. Given the sequence of $Y_m's$, this subsection aims to predict the $Y_{m+1}$ in the immediate future. The web server request access pattern, $Y_m$, can be shown to follow a particular time process, especially SARMA as explained later. After identifying underlying time series model, prediction is performed for $Y_{m+1}$.

Consider two time series $Y_1, Y_2, Y_3, \ldots$ and $Z_1, Z_2, Z_3, \ldots$ where the $Y$ series represents unpredictable shocks that affect the value of $Z$ series. A pure S-ARMA model of orders $P$ and $Q$, and $S$ seasonal periods (where correlation only occurs at lags that are multiples of seasonal periods) is $Y_t - \Phi_1 Y_{t-s} - \ldots - \Phi_P Y_{t-PS} = Z_t + \Theta_1 Z_{t-s} + \ldots + \Theta_Q Z_{t-QS}$. Rewriting the previous equation with backward shift operator $\beta$ the following expression can be derived, $\Phi(\beta^P)Y_t = \Theta(\beta^Q)Z_t$. The multiplicative seasonal-ARMA model (a combination of pure seasonal
ARMA model and regular ARMA) is used to fit most of the server access traces in this study. Regular ARMA of orders \((p, q)\) is 
\[ -\phi_1 Y_{t-1} - \phi_2 Y_{t-2} - \ldots - \phi_p Y_{t-p} = Z_t + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q}, \]
which with the backward shift operator is: 
\[ \phi(B^p)Y_t = \theta(B^q)Z_t. \]
With the pure seasonal ARMA of order \((P, Q)\), the multiplicative S-ARMA can be obtained as 
\[ \phi(B^p)\Phi(B^P)Y_t = \theta(B^q)\Theta(B^Q)Z_t. \]

The collections of \(Y_i\), mean and variance of arrival rate, are done in the following manner and shown in figure 4.3. For the \(i^{th}\) window, it consists of \(K\) unit intervals. The total number of requests, \(r_{ij}\), of \(j^{th}\) unit interval at \(i^{th}\) window is stored for calculating \(Y_i\). Eventually, \(Y_i\) is obtained through \(r_{i,1}, r_{i,2}, \ldots r_{i,K} \forall i\). The mechanism for deciding the window size has been detailed in the next subsection.

**Window size** To determine the proper window size, different criteria need to be considered and satisfied. These criteria are detailed below and supplemented with corresponding figure 4.3:

1. Independence of \(r_{i,1}, r_{i,2}, \ldots r_{i,K}\) within a window \(i\): The traffic parameters, \(Y_1, Y_2, \ldots Y_n\) are obtained and predicted in \(n^{th}\) window where the input data, \(r_{i,1}, r_{i,2}, \ldots r_{i,K}, \forall window\), pass the independence test. Run test, auto-correlation function and partial auto-correlation function are applied on all windows to check the independence of \(r_{i,1}, r_{i,2}, \ldots r_{i,K}\) for identifying window length. Note that the values of \(r_{i,1}, r_{i,2}, \ldots r_{i,K}\) are independent of length of window under testing. As the window size decreases, the number of windows at a fixed duration passing independence test increases. Hence, input data with window length below certain values, such as one hour, can always guarantee the independence of data.
input. Note that the fact that total requests collected in unit interval \( r_{i,1}, r_{i,2}, \ldots r_{i,K} \) pass independence test doesn’t imply the independence of inter-arrival times of each request within the window \( i \).

2. Accuracy of prediction: The window length providing more accurate prediction is preferable. Predictability of traffic parameters can be evaluated by i) prediction result directly and ii) performance analysis based on predicted values. \( R^2 \) related statistics and residual analysis are common practices for evaluating fitting and prediction results. Assuming steady state analytical performance functions well, it still requires time for underlying queueing system to reach its steady state. Therefore, the performance prediction from queueing model can indirectly decide the window length. A simulation system is usually constructed for this purpose. A big window can provide better fittings for traffic parameters and stability of system reaching steady state. On the other hand, a small window decreases the impact of collecting statistics by windows, especially at the presence of bursty arrivals and highly varying file sizes.

3. Practical issues: In addition to independence of input data and predictability, practical concerns, such as computational complexity and time of execution of control action, make this study consider a uniform window size, even though it cannot perfectly capture the exact transition of the traffic parameters.

Note that the actual window size and parameters in the SARMA models varies from trace to trace. In initial experiments on trial traces, it reveals that the window size between 30 minutes to an hour can perform well. Overall, this study suggests first to use sets of training data sets to obtain parameters off-line. Once the parameters and models have been identified, all the traffic parameters over the windows can be predicted by fixed numbers of historical moving windows. The implementation of prediction methodologies is done in an on-line fashion to provide the latest update of the values to improve the accuracy of the predictions. In order to minimize the
prediction errors, this study suggests to actually add a safe margin, such as mean plus standard deviation.

4.3.1.2 File size

Similar to arrival process, this subsection intends to build up a prediction model for mean and variance of file size over a time window for applying queuing performance analysis. File size is linearly related to the service time by scaling to server processing frequency, \( f \). As for the window size, it uses the choices of window size made by arrival process for the following reasons. Since the file size distribution only shows independent weak time-varying or no time varying pattern, the length of window is not that important for predicting file-size related parameters. Couple of fitting methods, such as ARMA, kinds of regression, and Winter’s smoothing, have been applied to predict the average and variance of file size across the windows. Because the average file size still shows light correlation among data points, Winter’s smoothing can do a relatively better job, compared with other methods. Due to the high variability of file size distribution across time, a safe margin, usually an estimated standard deviation is suggested and practised by this study to add to the prediction value in order to minimize the impact of underestimated prediction error.

For Winter’s smoothing [85], a prediction of \( Y_k \), the average file size of window \( k \), is obtained as

\[
Y_k = a\left(\frac{Y_k}{S_{k-p}}\right) + (1-a)(Y_{k-1} + T_{k-1}),
\]

for a constant \( a \), with

\[
T_k = b(Y_k - Y_{k-1}) + (1-b)T_{k-1},
\]

being the trend estimator where \( b \) is smoothing constant and

\[
S_k = c\left(\frac{Y_k}{Y_{k-1}}\right) + (1-c)S_{k-p}
\]

is an estimate of the season factor, with \( c \) being the seasonal smoothing constant and \( p \) being the season period. Note that the parameter values will be trace dependent.

Note that proposed prediction mechanism can function as input to obtain all types of performance measurement. The mean and variance of arrival rate and file size from the proposed methodology can be used as input for standard queuing performance, such as \( G/G/m \). This thesis adopts the proposed prediction mechanism for critical traffic parameters and then applies \( G/G/m \) as piecewise queueing stationary analysis to develop control mechanisms.
4.3.2 Control mechanisms

This study next proposes a non-linear queueing-based optimization mechanisms to determine the number of servers \( m_i \) allocated to each application \( i \) and their frequency \( f_i \) at any instant. It consists two parts: i) queueing performance analysis and ii) and non-linear optimization algorithm detailed in the following subsections. Further this study adopts feedback control base mechanism in [28] and further proposes a hybrid approach based on these two mechanisms as in subsection 4.3.2.3. Note that all \( m_i \) servers of application \( i \) run at the same frequency \( f_i \) as explained in section 4.1.

4.3.2.1 Queuing analysis

Following the prediction mechanism in previous subsection, mean arrival rate \( (\lambda_i) \), variance of arrival rate \( (\sigma^2_{\lambda_i}) \), mean file size \( (\phi_i) \) and variance of file size \( (\sigma^2_{\phi_i}) \) of each control interval are readily available. In order to obtain performance measures perform \( G/G/m \) analysis, additional information of squared coefficient of variation of request inter-arrival times \( (C^2_{\tau_i}) \), and squared coefficient of variation of file size \( (C^2_s) \) for each application \( i \) are required. It’s straightforward to obtain the expression of \( C^2_s \) by \( (\sigma^2_s) \). Squared coefficient of variation of request inter-arrival times is as following

\[
C^2_{\tau} = \frac{(\text{variance} - \text{interarrival - times})}{(\text{mean} - \text{interarrival - times})^2} = \frac{(\sigma^2_{\tau})}{\tau^2}
\]

Due to the limit of data collection process, only number of arrival is collected, but data of inter-arrival times is required for queueing analysis. This study therefore, derives variance-interarrival-times, \( (\sigma^2_{\tau}) \), by adopting the approximation in renewal theory [48] that number of customer arriving during the period of \( t \), \( N(t) \), has a normal distribution with mean of \( \frac{t}{\tau} \) and variance \( \frac{(\sigma^2_{\tau})}{(\tau)^2} \). Therefore, \( \sigma^2_{\tau} \) can be obtained backwardly by \( \sigma^2_{\lambda} \).

Further, using the predictions of \( \lambda(i) \), \( C^2_{\tau}(i) \), \( \phi(i) \), and \( C^2_s(i) \) for a given interval for each
application \( i \), an expression for the predicted average response times (\( W_i \)) in that given interval is obtained. Queue-\( i \) is approximated to be a \( G/G/m_i \) queue with independent identically distributed (i.i.d.) inter-arrival times and i.i.d. service times. Further, it’s assumed that the time interval is large enough that steady state results can be used (note that this is one of the reasons why results with this technique may not be very good for small time granularities of control as following evaluations section will show). There are several approximations for response time in the literature, of which many are empirical. This study uses the method in Bolch et al [18], which states that

\[
W_i = \frac{\phi(i)}{\beta \cdot f_i} + \frac{\alpha_{m_i} \phi(i)}{\beta \cdot f_i} \left( \frac{1}{1 - \rho_i} \right) \left( \frac{C_2^2(i) + C_s^2(i)}{2m_i} \right),
\]

(4.1)

where \( \beta \cdot f_i \) is the bandwidth of the server in bytes served per second for application \( i \) (\( \beta \) is a constant that is calculated by measuring the service time of HTTP requests of different sizes on an actual system), \( \rho_i = \frac{\lambda(i) \phi(i)}{m_i \cdot \beta \cdot f_i} \), and \( \alpha_{m_i} = \rho_i^{\frac{m_i + 1}{2}} \). Notice that \( W_i \) in Equation (4.1) is non-linear with respect to the decision/control variables \( f_i \) and \( m_i \). Also, there is no need to write down the constraint \( \rho_i < 1 \) explicitly in the optimization problem as satisfying the SLA constraint would automatically ensure this.

Note that \( W_i \) decreases with respect to both \( m_i \) and \( f_i \). Therefore the SLA constraint would be binding if \( f_i \) and \( m_i \) were continuous. However in the discrete case, an efficient frontier (i.e. for every \( f_i \) the smallest \( m_i \) rendering the SLA constraint feasible can be found) can be obtained. As there are \( \ell \) levels for \( f_i \), the proposed methodology will have to consider only \( \ell \) pairs of \( m_i \) and \( f_i \) for each application in each interval, as is exploited below.

### 4.3.2.2 Algorithm for the non-linear optimization

This study uses the following notation convention for developing solving algorithm. As shown in Figure 4.2, \( s \) and \( u \) are used to denote the corresponding subscripts for \( S \) and \( U \) intervals. For example, \( m_i(u) \) is the number of servers in queue-\( i \) at the \( u^{th} \) interval of duration \( T \), and
The frequency of a server in queue-i at the $s^{th}$ interval of duration $t$ for this given $u$.

The optimization problem can then be re-written in terms of the decision variables $f_i(u, s)$ and $m_i(u)$ (assume for all $i$, $m_i(0) = 0$) as:

$$
\min_{f_i(u, s), m_i(u)} \sum_{u=1}^{U} \left( \sum_{s=1}^{S} \sum_{i=1}^{N} K_s m_i(u) (P_{fixed} + P_f * f_i(u, s)^3) t \right) + B_o \left[ \sum_{i=1}^{N} m_i(u) - \sum_{i=1}^{N} m_i(u-1) \right]^+ 
$$

subject to

$$
W_i \leq \bar{W}_i \text{ for all } i \n$$

$$
\sum_{i=1}^{N} m_i(u) \leq M \text{ for all } u, \text{ and } m_i(u) \text{ is a non-negative integer} \n$$

$$f_i \in \mathcal{F} \n$$

$$m_i(u) \in \mathcal{I}^+, \forall u \in [1, N] \n$$

Clearly, the above optimization is non-linear and discrete in terms of the decision variables for both the objective and the constraints. Moreover, it appears as a NP hard problem that is beyond the handling of standard optimization techniques. Since only a finite number of frequency alternatives is considered, it is tempting to do a complete enumeration to determine the optimal solution. However, a complete enumeration would require comparing $O(\ell^S M_U N)$ values - and performing these periodically during execution. Therefore a heuristic is resorted as explained below.

The prediction model described above is dynamically revised, based on the previous observation (for $u - 1$), and then run the following two steps at the beginning of each $u$. The steps are explained as following. The case $t = T$ is first considered, and therefore for all intervals $s = 1$. Under this framework, $f_i(u, s) = f_i(u)$ is used for ease of notation.

**Step 1:** The proposed method first obtains a feasible solution that would result in an upper
bound to the objective function. Only a single interval and optimize the parameters \( f_i(u) \) and \( m_i(u) \) are considered for the simplified objective as below:

\[
\min_{f_i(u), m_i(u)} \sum_{i=1}^{N} K g m_i(u)(P_{\text{fixed}} + P_f * f_i(u)^3) t + B_0 \sum_{i=1}^{N} m_i(u)
\]

There are two ways of solving this, one is to assume the decision variables \( (m_i(u) \) and \( f_i(u) \)) are continuous and use standard non-linear programming techniques. This is especially useful when the problem is scaled to a large number of applications. However, this study uses a second method where the proposed methodology exploits the monotonicity properties of the objective function and constraints as well as the fact that the frequencies take only a small number of discrete values. For that purpose, estimates of moments of inter-arrival times and file sizes are used, across all applications and across all intervals. Then, for each interval \( u \) the proposed methodology starts by finding the minimum number of servers \( m_i(u) \) for all applications \( i \) so that the constraint \( W_i(u) \leq W_i \) can be satisfied using the highest frequency for the servers. Note that the total number of servers in the hosting center \( M \) is large enough that this would guarantee the constraint \( \sum_{i=1}^{N} m_i(u) \leq M \) to be automatically satisfied - otherwise it implies the SLA has been chosen poorly since it would have been violated even without any energy management. This solution can be improved by recognizing that the objective is a cubic in terms of the frequency that needs to be reduced. Therefore by suitably decreasing \( f_i(u) \) and increasing \( m_i(u) \), as long as the constraints are satisfied, a solution that would be close to the optimal solution can be obtained. For example, the proposed algorithm is trying to find the number of servers that are needed when they are all operating at lower frequencies that can give the most power savings. When reducing \( f_i(u) \), applications are selected in decreasing order of \( f_i(u) \) for each interval.

**Step 2:** The proposed algorithm next considers all the intervals together. The actual objective function that is the total cost across the entire execution, is used for this purpose. The upper
bound solution from the previous step is used and work the way accepting feasible solutions that reduce the objective function value. If a point in this feasible space improves the objective, solution is accept and it searches further in its neighborhood. In the proposed approach considers one interval at a time (going from the first to the last) and in each interval the applications is selected in the decreasing order of their frequencies. For each application the number of servers in that interval is compared against those of the next interval. The algorithm tries to level off the number of servers (to the extent possible) whenever the resulting solution improves. It searches greedily giving importance to the points where the number of servers for this interval are close to the number for the previous interval (to reduce turn-on costs). Thereafter, the frequencies is tuned so that the resulting solution remains feasible.

Note that \( T = t \) is being assumed in the above two steps. To handle the cases where \( T > t \), the proposed algorithm first performs the above optimization (assuming \( t = T \) and using average values of arrival and file size parameters over every interval) at each of \( u = 1 \) to \( U \), to first determine the \( m_i(u) \). Subsequently, using the prediction information for each \( s \), the appropriate \( f_i(u, s) \) is determined by using this pre-determined value of \( m_i(u) \).

### 4.3.2.3 Control theoretic approach and hybrid approach

The SSQN approach, described above, can be viewed to be “pro-active”, since it predicts workload behavior for the near future and tunes server allocation and frequency setting appropriately. While it can do well when the predictions are fairly accurate, and the steady state stochastic behavior is obeyed (requiring a fairly coarse-grained window), it may not be adaptive enough for fine-grained transient behavior. On the other hand, the control theoretic approach proposed by [27] can be viewed to be “reactive” since it bases its decisions on feedback. The evaluation of control theocratical approach is thus better at finer granularities, though it can possibly miss out on the predictive information for a longer granularity of optimization. This leads this study to consider a hybrid scheme, where the predictive information of the SQNO approach is used to
determine server allocation (section ??) at a granularity of $T$, and the feedback-based control theoretic approach is applied for frequency setting at the smaller granularity of $t$. Note that this strategy is also in agreement with underlying philosophy where server provisioning costs (in terms of time for server turn-on and application migration) are anticipated to be much higher than frequency control, and are thus expected to be done less frequently (i.e. $T > t$).

4.4 Experiments

4.4.1 Experimental set up

This study uses real HTTP traces obtained from [4] during the last week of September and early October 2004, for the evaluation, whose arrival time characteristics are shown in Figure 4.4. There are 3 traces in all, denoted as Applications 1 to 3, with each trace being for a 3 day duration. The data of the first 2 days (called training data) are used to build the prediction model for the SQNO approach and the system identification model for the control-theoretic approach, while only the 3rd day’s trace is used in the actual simulation/evaluation. These three applications are mixed in the experiments to get different workloads, WL1 to WL3: WL1 contains only one application, namely Application 2; WL2 includes both Applications 1 and 2; while WL3 includes all 3 applications.

The techniques have been evaluated using a simulator built on top of the CSIM simulation software. The simulator takes the server frequency $f_i$ and the number of servers $m_i$ for each time period as inputs, and generates response time $W_i$ as system output for a given static HTTP request trace. During the simulation, it also calculates the energy consumed, number of reboots, and cost of operation. The cost of reboots and migrating a server from one application to another are inputs to the simulator.

Since the primary interest of this study lies on the CPU power, which as explained in section 2.1 dominates over the other components, this study assumes that the static HTTP requests hit
in the cache, i.e. there are no disk accesses. Further, as pointed out in [33], 99% of the HTTP requests can be easily served from the cache.

The simulator for the hosting center models a number of these independent servers each with independent frequency modulation, and each being a FCFS queuing system. Then there is only a need to model the parameters for (i) service times of different requests at each frequency, and (ii) energy consumption when a server spent a specified amount of time at a certain frequency. Server class processors with DVS are only starting to arrive in the market, and as of this writing this study does not have such a platform for experiments. In fact, most previous work in the area
(e.g. [33, 17]) extrapolated data from mobile/laptop processors (e.g. Transmeta). The proposed approach has been somewhat similar:

- This study ran microbenchmarks of requests with different file sizes on a Xeon server machine which hit in the cache and used this as the service times for the requests at the highest operating frequency (2.6 GHz) of the processor. As expected, the relationship between file sizes and service times were more or less linear which can be seen from Figure 4.5 (shows this relationship for a number of files of different sizes and respective service-times).

- Similar experiments were also conducted on a laptop with DVS capabilities to confirm that the service time is inversely proportional to the operating frequency of the CPU. Table 4.1 shows this relationship for a number of different frequencies and respective service-times for a 256 KB file. This study then appropriately adjusted the service times in our simulation model for the server class CPU. Each node in the simulated cluster uses these service rate (the slope shown in Figure 4.5 in serving requests in FCFS order.

- For the power model, the data sheets are directly used to model the power value at the highest frequency. The maximum power consumption at the highest frequency in our simulator is 100 W, which roughly matches the consumption on current server CPUs. This study then uses the well-known cubic relationship between frequency and power, to model the power at each discrete frequency, which is similar to the technique in [33].

<table>
<thead>
<tr>
<th>Frequency (MHz)</th>
<th>Service time (millisecond)</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td>1.12</td>
</tr>
<tr>
<td>800</td>
<td>0.83</td>
</tr>
<tr>
<td>1000</td>
<td>0.69</td>
</tr>
<tr>
<td>1200</td>
<td>0.55</td>
</tr>
<tr>
<td>1400</td>
<td>0.47</td>
</tr>
<tr>
<td>1600</td>
<td>0.42</td>
</tr>
</tbody>
</table>

Table 4.1. For a 256 KB file, service-times at different operating frequencies
This study restricts the results presented here to the parameter values shown in Table A.1 in the interest of space. The table gives the different possible CPU frequencies, and their associated power consumption values. The electricity cost ($K_3$) is close to that being charged currently, and the dollar cost associated with wear-and-tear per reboot is based approximately on the cost of a disk replacement (say $200 including personnel) and the rated MTBF (say 40000 start-stop cycles). Though our model allows per application $\bar{W}_i$, in the interest of clarity, this chapter uses the same target response time for all applications of 6 ms. It have been ensured that this is achievable for each workload by provisioning an appropriate number of servers ($M$) as given in the table. This chapter considers different combinations of $T$ and $t$, with $t < T$, i.e. DVS being done much more frequently than server allocation, with $(T,t)$ being used to denote the experiment.

**Metrics:** The main statistics presented here include (i) the objective function (i.e. Cost), (ii) the energy consumption (Energy %) expressed as a percentage of the energy that would be consumed if all the servers were running all the time at the highest frequency, i.e. without any
Table 4.2. Default parameter values for simulation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{F}$ (in GHz)</td>
<td>(1.4, 1.57, 1.74, 1.91, 2.08, 2.25, 2.42, 2.6)</td>
</tr>
<tr>
<td>$P$ (in Watts for each $f$)</td>
<td>(60, 63, 66.8, 71.3, 76.8, 83.2, 90.7, 100)</td>
</tr>
<tr>
<td>$K_3$ (cents/KWH)</td>
<td>10</td>
</tr>
<tr>
<td>$C_r$ (cents/boot)</td>
<td>20,000/40,000 = 0.5</td>
</tr>
<tr>
<td>$B_o$ (cents/boot)</td>
<td>$C_r + 0.025 = 0.525$</td>
</tr>
<tr>
<td>$T_{\text{reboot}}, T_{\text{migrate}}$</td>
<td>90 secs, 20 secs</td>
</tr>
<tr>
<td>$M$</td>
<td>13 (for WL1), 31 (for WL2), 39 (for WL3)</td>
</tr>
<tr>
<td>$W_{i,T}$ (in msec)</td>
<td>6</td>
</tr>
<tr>
<td>$T$ (in minutes)</td>
<td>(60, 20, 10)</td>
</tr>
<tr>
<td>$t$ (in minutes)</td>
<td>(60, 20, 10, 5) with $t \leq T$</td>
</tr>
</tbody>
</table>

power management, (iii) the number of reboots ($\#\text{reboots}$), and (iv) the mean response time ($RT$). When evaluating any scheme this study wants to first check whether it is able to meet the response time SLA (i.e. a target of 6 ms), before checking its energy consumption or cost. However, it should be noted that some schemes may very marginally violate this SLA while saving substantial cost. In order to not penalize such a scheme, this study allows a 10% slack in the SLA, i.e. it considers schemes that can go as high as 6.6 ms in its mean response time when comparing them, and this study terms such a scheme to be a viable option.

4.4.2 Results

4.4.2.1 Need for sophisticated strategies

Before evaluating the proposed schemes, this study first presents some results to motivate their need, by examining the following three “simple” schemes running WL1:

1. Fixed servers, constant frequency during the entire experiment: Results with such a scheme for the maximum servers (=13) by running the experiment in a brute-force fashion with each frequency level is shown in Table 5.3 (a). Due to the stochastic nature of the problem, this study considers 10% tolerance margin when converting it to a deterministic problem through prediction. That means if any “viable” option with this scheme (i.e. $RT < 6.6$) is demanded, the lowest cost can be obtained is 196 cents, which is around 15% higher than
what SQNO can provide at (60,60) - see Table 5.4 - and the proposed approach can do even better at other granularities.

2. **Highest frequency, constant number of servers during the entire experiment:** This corresponds to statically configuring the hosting center with the minimum number of servers needed to meet the SLA if they were all to run at the maximum speed. Looking at the results for this scheme in Table 5.3 (b), it takes at least 8 servers operating at full frequency, before this becomes a “viable” option. Even at this number of servers, the energy consumption and cost are higher than for our schemes at different time granularities (compare with Table 5.4).

3. **Maximum number of servers, switch between \( f_{\text{min}} \) and \( f_{\text{max}} \):** Whenever there is no request, a server switches to the minimum frequency (no cost is assumed for such switching), and whenever a request arrives it switches to the maximum frequency until it has finished serving the request. Clearly, the response time is not any different from operating at maximum frequency. However, the energy (72.9% of having them all operate at highest frequency) and cost (227.55 cents) are much higher than what we can get with our schemes.

Note that the first two schemes require a brute-force evaluation of all alternatives, and are thus not suitable in practice. The point in showing their results was to point out that even if one had a very good static server capacity allocation mechanism for the hosting center, results with a dynamic power management strategy can provide better savings due to time-varying behavior of the workload. One could envision the above third scheme to be a “no-brainer” power management strategy if there are no costs to transition between frequencies. However, with reasonable load conditions, there is not that much scope for operating at lower frequencies. The reader should note that any of our three proposals could be used in conjunction with this third simple mechanism in order to set the maximum frequency at which to serve a request dynamically (and to otherwise bring it down to \( f_{\text{min}} \)). However, that is not considered in the evaluations.
Table 4.3. Results for the “simple” strategies

<table>
<thead>
<tr>
<th>Frequency (GHz), #</th>
<th>Cost (cents)</th>
<th>RT (ms)</th>
<th>Energy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1.40, 13)</td>
<td>187.20</td>
<td>9.08</td>
<td>60.00</td>
</tr>
<tr>
<td>(1.57, 13)</td>
<td>196.76</td>
<td>5.07</td>
<td>63.07</td>
</tr>
<tr>
<td>(1.74, 13)</td>
<td>208.66</td>
<td>3.25</td>
<td>66.88</td>
</tr>
<tr>
<td>(1.91, 13)</td>
<td>223.14</td>
<td>2.34</td>
<td>71.52</td>
</tr>
<tr>
<td>(2.08, 13)</td>
<td>240.46</td>
<td>1.87</td>
<td>77.07</td>
</tr>
<tr>
<td>(2.25, 13)</td>
<td>260.87</td>
<td>1.57</td>
<td>83.61</td>
</tr>
<tr>
<td>(2.42, 13)</td>
<td>284.63</td>
<td>1.36</td>
<td>91.23</td>
</tr>
<tr>
<td>(2.60, 13)</td>
<td>312.00</td>
<td>1.20</td>
<td>100.00</td>
</tr>
</tbody>
</table>

a) Fixed servers = 13, const. freq.

<table>
<thead>
<tr>
<th>Frequency (GHz), #</th>
<th>Cost (cents)</th>
<th>RT (ms)</th>
<th>Energy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2.60, 1)</td>
<td>24.00</td>
<td>29368.20</td>
<td>7.69</td>
</tr>
<tr>
<td>(2.60, 2)</td>
<td>48.00</td>
<td>17846.13</td>
<td>15.38</td>
</tr>
<tr>
<td>(2.60, 4)</td>
<td>96.00</td>
<td>3065.25</td>
<td>30.77</td>
</tr>
<tr>
<td>(2.60, 6)</td>
<td>144.00</td>
<td>28.45</td>
<td>46.15</td>
</tr>
<tr>
<td>(2.60, 8)</td>
<td>192.00</td>
<td>5.05</td>
<td>61.54</td>
</tr>
<tr>
<td>(2.60, 10)</td>
<td>240.00</td>
<td>1.95</td>
<td>76.92</td>
</tr>
<tr>
<td>(2.60, 12)</td>
<td>288.00</td>
<td>1.35</td>
<td>92.31</td>
</tr>
<tr>
<td>(2.60, 13)</td>
<td>312.00</td>
<td>1.20</td>
<td>100.00</td>
</tr>
</tbody>
</table>

(b) Max. freq. = 2.6 GHz, const. servers

4.4.2.2 Comparison with IBM method

The most closely related power management mechanism from IBM [33] uses a rather simple feedback control mechanism to determine the frequency and number of servers for the next interval given those of the current interval and the observed utilization/response time, and these two mechanisms are integrated (i.e. $T = t$). It is a purely feedback mechanism and does not use any prediction, and it is not intended to manage server allocation across applications. In the interest of fairness, this study should mainly compare results for WL1 (the single application workload in Table 5.4), and even there it’s seen that while their method gives better energy savings (around 70%), it is not “viable” (i.e. the response times are never meeting the required SLA).

Since their method treats response time on a best effort basis, rather than a constraint to obey, energy optimization takes center-stage, leading to much worse response times. As mentioned earlier, the SLA is usually much more important (the revenue earner) to the hosting center. Our
### 4.4.2.3 Comparing the proposed schemes

Table 5.4 can be used to compare our three strategies for the three workloads. This study mainly focuses on the “viable” executions, i.e. those where the average response time is less than 6.6 ms.

As described earlier, the pro-active SQNO approach uses predictions to anticipate workload behavior and steady-state analysis, both of which need larger time granularities for better accuracy. At these larger granularities, the Control theoretic approach does not have any prediction information to optimize for the next interval, using feedback from the last interval that is again coarse-grained. Consequently, it’s seen that the SQNO approach does much better than the re-

---

**Table 4.4.** Comparison between schemes on savings of electricity-cost

<table>
<thead>
<tr>
<th>Schemes</th>
<th>WL1</th>
<th>WL2</th>
<th>WL3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost (cents)</td>
<td>Egy (%)</td>
<td>rb</td>
</tr>
<tr>
<td>Queue</td>
<td>45.80</td>
<td>25.20</td>
<td>62.31</td>
</tr>
<tr>
<td>IBM</td>
<td>10.69</td>
<td>10.69</td>
<td>7.09</td>
</tr>
<tr>
<td></td>
<td>45.80</td>
<td>25.20</td>
<td>62.31</td>
</tr>
</tbody>
</table>

---

Schemes are able to meet the SLA in many cases, while still providing reasonable energy savings (around 50%).
active Control scheme at large \((T,t)\) granularities. For example, at \((60,60)\), the proposed SQNO is around 10% better in cost than Control for WL1.

To understand the execution characteristics of these experiments at large \((T,t)\), An example
is given about execution of Application 3 in WL3 in terms of the number of servers provisioned ($m_i$), their frequency ($f_i$) and the corresponding response time ($W_i$) over time in Figure 4.6 (b).

If the response time behavior of Control (60,60) is examined, there is a spike at hour 7. This is because there was a load burst in this hour, which the mechanism could not detect at the end of hour 6 (because there is only feedback). On the other hand, SQNO has advance knowledge of the workload for the next hour, and is able to make better judgments on server allocation. This also reflects on the corresponding frequency that is being set, resulting in the lower cost for SQNO.

When the focus moves to the other extreme of small time granularities (say (10,5)), it’s seen that the inaccuracies of the SQNO strategy significantly impact its viability, and this option does not meet the SLA in any of the 3 workloads. However, by getting very frequent feedback from the system, the reactive Control scheme is able to perform a better job of server allocation and frequency modulation to lower the costs.

The result of inaccuracies of SQNO at small ($T, t$) is obvious in Figure 4.6 (a), which shows the time varying behavior for Application 2 in WL3. The SQNO scheme appears to under-estimate the required capacity, thus turning down more servers than Control. This results in a higher response time.

The hybrid scheme that is intended to benefit from the merits of these two schemes, does give a cost between these two at either extremes of time granularities. When the large $T$ and short $t$ ranges (e.g. (60,5)) is evaluated, hybrid benefits from the pro-active mechanism for server allocation from the SQNO strategy (it uses the same number of servers as SQNO), and the reactive feedback from the underlying system to set frequencies based on short term variations to meet the response time SLA.
4.4.3 Varying the parameters

One would be interested in finding out how these schemes scale as moving to a larger environment with a lot more servers that host more applications. This study have conducted an experiment with \( M = 285 \) servers running \( N = 30 \) applications. Since it is difficult to procure so many different traces, the three traces used earlier are used and are replicated themselves by using randomized phase differences to synthetically generate workloads. Results for this workload are given in Table 4.5 for representative \((T, t)\) granularities, viz., (60,60), (60,5) and (10,5). As before, the SQNO approach is more viable at the larger time granularity, and the inaccuracies cause a significant degradation in response times at the finer granularity. In fact, these inaccuracies at fine granularities result in poor choice of server allocation in Hybrid at (10, 5). However, at the (60, 5) granularity, Hybrid is again able to benefit from the pro-active prediction of SQNO for server allocation, and the feedback of Control for frequency modulation, giving the lowest response times.

<table>
<thead>
<tr>
<th>((T, t))</th>
<th>Schemes</th>
<th>Cost (cents)</th>
<th>RT (ms)</th>
<th>#reboots</th>
</tr>
</thead>
<tbody>
<tr>
<td>(60, 60)</td>
<td>Queueing</td>
<td>2625.2</td>
<td>5.5</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>Control</td>
<td>2679.8</td>
<td>24.1</td>
<td>230</td>
</tr>
<tr>
<td></td>
<td>Hybrid</td>
<td>2593.2</td>
<td>9.6</td>
<td>275</td>
</tr>
<tr>
<td>(60, 5)</td>
<td>Queueing</td>
<td>2465.1</td>
<td>35.4</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>Control</td>
<td>2427.8</td>
<td>13.8</td>
<td>214</td>
</tr>
<tr>
<td></td>
<td>Hybrid</td>
<td>2514.8</td>
<td>6.6</td>
<td>275</td>
</tr>
<tr>
<td>(10, 5)</td>
<td>Queueing</td>
<td>2435.5</td>
<td>73.4</td>
<td>342</td>
</tr>
<tr>
<td></td>
<td>Control</td>
<td>2530.5</td>
<td>7.3</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>Hybrid</td>
<td>2442.3</td>
<td>31.1</td>
<td>342</td>
</tr>
</tbody>
</table>

Table 4.5. Results for workload with 30 applications and 285 servers

Note that by modulating parameters of the system model, the proposed approach, SQNO, can be targeted at a wide spectrum of systems, operating conditions and cost functions. For instance, setting \( T \) to \( \infty \) implies conducting server provisioning once (i.e. a static configuration of the number of servers), and then managing power only with DVS. Similarly, setting \( t \) to \( \infty \) implies that only server turn off/on is employed for power management. This study has also
conducted experiments varying the dollar ($B_o$) and time ($T_{reboot}$) of reboots ($B_o$), and the time for application migration ($T_{migrate}$). The results of those experiments are as follows:

- **Varying $B_o$**: One can study the impact of server turn off/on, on the objective function. For instance, one can completely ignore boot up costs (whether it be the associated energy, or the wear-and-tear) by setting $B_o = 0$. This study also considers the case where the wear-and-tear is on the entire machine, and not just on the disk (as was done in the previous experiments) by increasing $C_r$ ten times (assuming a $2,000$ cluster node and $40,000$ start-stop cycle MTBF) to give $B_o = 5.025$. Representative results are given for the $B_o = 0$ experiments in Table 4.6. As can be expected, since there are lower costs for server turn off/on, the overall costs have decreased for all schemes (compared to Table 5.4). These results also re-affirm the earlier observations of SQNO incurring a lower cost at larger time granularities, and Control being more effective at smaller granularities.

<table>
<thead>
<tr>
<th>$(T,t)$</th>
<th>Schemes</th>
<th>Cost (cents)</th>
<th>RT (ms)</th>
<th>Energy (%)</th>
<th>#reboots</th>
</tr>
</thead>
<tbody>
<tr>
<td>(60, 60)</td>
<td>Queueing</td>
<td>√</td>
<td>469.48</td>
<td>5.76</td>
<td>50.26</td>
</tr>
<tr>
<td></td>
<td>Control</td>
<td></td>
<td>491.53</td>
<td>15.53</td>
<td>52.56</td>
</tr>
<tr>
<td></td>
<td>Hybrid</td>
<td></td>
<td>485.11</td>
<td>10.28</td>
<td>51.79</td>
</tr>
<tr>
<td>(60, 5)</td>
<td>Queueing</td>
<td>√</td>
<td>450.69</td>
<td>8.15</td>
<td>48.21</td>
</tr>
<tr>
<td></td>
<td>Control</td>
<td></td>
<td>460.35</td>
<td>6.53</td>
<td>49.23</td>
</tr>
<tr>
<td></td>
<td>Hybrid</td>
<td></td>
<td>466.25</td>
<td>5.85</td>
<td>49.74</td>
</tr>
<tr>
<td>(10, 5)</td>
<td>Queueing</td>
<td></td>
<td>430.20</td>
<td>19.81</td>
<td>45.90</td>
</tr>
<tr>
<td></td>
<td>Control</td>
<td></td>
<td>√</td>
<td>461.63</td>
<td>6.60</td>
</tr>
<tr>
<td></td>
<td>Hybrid</td>
<td></td>
<td></td>
<td></td>
<td>432.08</td>
</tr>
</tbody>
</table>

*Table 4.6. Results for $B_o = 0$, WL3*

Further, this study ignores the current objective function, set $\mathcal{F} = f_{max}$, and find the minimum number of servers needed to meet $W_i$ SLA for each application at any time. This is the traditional dynamic server provisioning problem without considering energy consumption. Such an experiment has been conducted for WL3, and it’s found that the number of servers that are needed in all is 29 where the $m_i$ keeps changing over time (at 60 minute granularity using SQNO). This is lower than $11 + 11 + 11 = 33$ servers that would be needed if each of the applications
were to be run in isolation, i.e. the server farm is partitioned between the applications to ensure SLA for each. This is because the peak demands across applications are not necessarily coming at the same time. Note that the proposed methodology is able to optimize energy by considering server provisioning and DVS simultaneously. On the other hand, schemes such as [33], rely on a higher level provisioning mechanism after which they can perform their energy optimizations.

4.5 Concluding remarks

This study has presented the first formalism to the problem of reducing server energy consumption at hosting centers running multiple applications towards the goal of meeting performance based SLAs to client requests. Though prior studies have shown energy savings for server clusters by server turn-offs and DVS, these savings come at a rather expensive cost in terms of violating the performance-based SLAs (which are extremely important to maintain the revenue stream). Further, previous proposals have not considered the cost of server turn-offs, not just in terms of time overheads, but also in the wear-and-tear of components over an extended period of time.

The proposed solution strategies couple (i) server provisioning for different applications, and (ii) DVS towards enhancing power savings, while still allowing different mechanisms for achieving these two actions, unlike any prior work. Two new solution strategies have been presented to this problem. The first is a pro-active solution (SQNO) which predicts workload behavior for the near future and uses this information to conduct non-linear optimization based on steady state queuing analysis. This study also implemented reactive feedback control solution [27]. While SQNO can be a better alternative than Control when the workload behavior for the near future is different from the recent past, the steady state assumptions may not hold over shorter granularities. Feedback Control is preferable at these shorter time granularities, but the information obtained at the last monitoring period may not be very recent at larger time granularities. The Hybrid strategy, on the other hand, can use the predictive information of
SQNO at coarse time granularities for server provisioning, and the feedback of Control at short granularities for DVS. This Hybrid scheme is also well suited for a more practical setting where one would like to perform server provisioning less frequently (due to the high time overhead of bringing up servers or migrating applications), and perform DVS control more frequently. The good performance of feedback-based DVS in small granularity leads this study to develop a feedback based DVS utilizing the transience of workload in the following chapters.
There are two growingly important trends in today’s enterprises’ Information Technology (IT) management. Note that Enterprises are referred to as applications in chapter 4 in this study. On the one hand, the cost and complexity of system tuning and management is leading numerous enterprises to offload their IT demands/services to hosting/data centers. The other trend is the growing popularity of the Server-Based/Thin-Client Computing model, the computing power resides mainly at the back-end servers, with the clients serving as simple interface engines to the user. Both of trends lead enterprisers to adopt a centralized IT management of Server-Based computing, stemming from advantages in lowering Total Cost of Ownership - particularly in support and maintenance. This server-based model can also enhance system reliability and availability, and improve data security with higher resilience typically provisioned at the high-end servers. Moreover, this model automatically facilitates the sharing of hardware and software resources, making it a natural way of exploiting temporal (over time) and spatial (across the clients/services) heterogeneity of resource requirements across clients.

Back-end servers for an enterprise/application can sustain 1) various types of services, such as
mail services, database service, scientific computing services etc., and 2) all client users within the enterprise. As pointed out by [44, 73], it is beneficial to process the same type of services at the same node of servers, which usually have enough capacity to host more than one type of services. Evolving virtualization innovations [78, 3] are making it easier to seamlessly co-locate services with diverse hardware/software requirements on the back-end servers. When consolidating these services, one needs to be careful in co-locating them on the back-end servers. A poor packing of these services on the servers can cause undue interference between them, leading to possibly significant performance degradation in their execution (even with the presence of sufficient network bandwidth).

As mentioned in chapter 4, a growingly daunting cost of operation for these high-end systems is the power consumption. This problem is likely to get worse with increasing power densities, and heat dissipation has started limiting the frontiers to which performance can be pushed. One reason why this study may not aim to provide the aggregate maximum needs of all service/client sessions at the back-end all the time is that these maximum demands may not all come at the same time. Consequently, dynamic provisioning of servers, together with a dynamic assignment of client sessions to these servers, may be a more cost-effective option than always provisioning the maximum demands.

Driven by these motivations, this chapter addresses the problem of resource provisioning at the back-end servers to handle multiple service/client sessions, towards meeting performance goals while lowering the power consumption of these servers. The specific contributions made by this chapter are the following: (a) This study presents the first formulation of this problem as a constrained optimization problem, with the goals being energy minimization subject to the constraints of providing the capacity demands. One of the features of this formulation is a capacity assurance parameter \( \eta \) which is the percentile of total demand that strive to provide, rather than the maximum demands. This tunable parameter allows this study to trade-off any marginal performance benefits that one may get by provisioning 100% of the resource demands,
This study provides a two step solution strategy to this problem. In the first step, this study provisions the required number of servers and packs the client sessions on them. The novelty of our solution is in posing this as a bin-packing problem with the goal of balancing the distribution of load (based on mean and variance) across the servers. There are also several choices this study proposes for achieving this balance. After packing the client sessions, the second step uses a decentralized pro-active algorithm to set the operating frequency (DVS) for the near future to reduce energy consumption, while still meeting performance goals.
5.1 System model

Before getting into the specifics of how to take a number of services/client workloads and pack co-locate them on back-end servers, this study first gives details on the server and client/service models. This study also presents real workload data of clients and the case of co-locating them. Further, the associated metrics and overheads are discussed in the last subsection.

5.1.1 Servers

At the back-end, this study assumes a number of identical servers, each of which is capable of hosting any client’s workload sent to it. These server machines maintain full persistent states of user sessions. Note that despite hardware and operating system heterogeneities between the clients and the servers, virtualization technologies are available for such seamless offloading and subsequent migration. Many of today’s servers are being physically consolidated in a dense rack, which makes it easier for management together with advantages of saving space and being more cost-effective. These are typically referred to as blade systems.

Power management of these servers is gaining prominence not just because of the electricity cost for their operation, but also in the cost of cooling these systems. The CPU has been noted [49] to be the biggest contributor to this power, especially in densely packed dual processor blades, and there are two broad solution strategies from the software perspective to deal with this issue. One option is to completely power down the system, after migrating its load to other nodes [66]. Even if one is not completely powering down the system, which can have adverse consequences such as increasing the wear-and-tear of components (such as the disk) and having higher transition latencies, there exist different sleep states (e.g. the C1, C2 and C3 states in the ACPI [5] specifications) to which the CPU can transition when not executing. The other option is to use Dynamic Voltage Scaling (DVS), where the frequency can be slowed down (and voltage lowered) to consume lower power even when the CPU is active. DVS and CPU power
mode modulation can be performed at a rather fine granularity (milliseconds/seconds) due to negligible overheads, while machine turn on/off demands a coarser granularity of control (of the order of an hour or larger) since the overheads can run to a couple of minutes.

5.1.2 Services/Clients

Services sent to back-end servers from an application could be types of static http requests, dynamic pages update, interactive financial transaction, streaming of conference calls, and etc. Web traces available at today’s public domain are limited to static http requests, that are the just the small portions of all request pool. This study envisions the clients, whose workload being offloading to the back-end, to be running different applications ranging from office productivity tools to multimedia/graphics and other desktop applications. The system schematics of this chapter is shown in figure 5.1. Workloads merged from types services show different characteristics from ones merged from client sessions. For allotted servers of enterprises/application, workloads either sent from types services or clients are essentially interpreted as resources requirements, such as CPU, memory, or disk. Therefore, this chapters aims to propose a general methodology utilizing workload statistical characteristics, instead of attributes of workloads. Due to availability of services type of data from application at web hosting centers, this study mainly use the the client workload for motivating and evaluating of developed methodology. The client workload, and the later evaluations in this chapter, are specifically drawn from real-world data of load information of over 500 client user sessions in a large Fortune-500 Financial Institution that has been made available to this study. These sessions have been individually logged at a 15 second granularity with the following information available for every 15 seconds: (a) Average CPU utilization, (b) Average virtual memory pages/sec read from or written to disk to resolve hard page faults, and (c) Average disk queue length. With such aggregate resource usage information from each client, the goal is to meet these resource requirements during each time interval at the back-end servers.
5.1.3 Client workloads: a case for co-location

With consolidation, it not only brings the required computational power into a smaller physical space, but also reduce (relative to the number of clients) the number of back-end servers hosting these service workloads. Note that the peak resource requirements of all services are not always coming at the same time. The load variations - over time for a single service, and across services at a given time - can allow us to co-locate the workloads of several services on a single
server and still meet the resource demands.

To illustrate this observation, in Figures 5.2 (a), (b) and (c) it’s shown the raw information of the logged trace of the three resources (CPU, memory and disk) drawn at an hourly granularity for clarity over the 48 hour period. In each graph, the usage for a representative high, medium and low load client is shown to illustrate the variances across the clients. It’s also observed by this study that that despite some variances, the resource requirements for memory and disk are rather low across all the clients. Since these are not very intensive server workloads, their usage is not very demanding on the underlying system and can be easily met despite co-location for the ratio of clients to servers (less than 10:1) that it’s found later in this paper. The CPU is also the most power consuming of the system resources under consideration. Consequently, the CPU resource takes more prominence in the co-location problem (showing more variance and higher loads), and this resource is primal focus on for the rest of this chapter.

As it can be seen, the resource requirements - especially for the CPU - shows both space (across clients) varying and time-varying behavior:

- At any specific point in time, it’s found client with close to zero CPU utilization, as well as clients with nearly 100% utilization. This high spatial variance suggests that even if this study is to assume that the server to which this study is offloading is only as powerful as the client, then more than one client’s workload could be co-located/packed on a server and still be able to accommodate the resource demands. To illustrate this, in Figures 5.2 (d), (e) and (f), the resource requirements of all clients are cumulated by summing up their individual needs. Even assuming a server to be only as powerful as a client, it’s noticed that less than 15 servers (assuming the workload is completely divisible) can meet the peak resource demands for the depicted graph. Note that these graphs show averages over an hour, and there could be spikes within an hour exceeding these depicted values.

- As the graphs illustrate, the resource demands also vary over time. The time-varying
resource demands mandate that this study needs to dynamically modulate (i) the number of servers handling these client workloads (this study does not want to keep them on all the time for power considerations), (ii) the allocation/co-location of client workloads to a server, and (iii) the frequency (DVS) to provide the required capacity in a power efficient manner. These three resource modulation functionalities are employed by the techniques proposed in this study.

5.1.4 Metrics and overheads

This study mainly focuses on the resource allocation problem at the back-end servers to provide the necessary service workload CPU demands at the granularity/resolution of logging. The goal when performing the CPU resource allocation is to minimize the number of times the system is not able to provide the required amount of CPU capacity to a service session, in a 15 second window. Whenever the requisite capacity is not available to the session, this chapter terms it to be a spill-over since the deficiency in the capacity needs to be made up in the next few time windows. Consequently, this study uses the percentage of windows where there are spill-overs as a metric for comparison.

While reducing spill-overs is the primary goal, reducing the energy consumed by the servers is another desirable criterion. As stated earlier, energy consumption is a function of the number of operating servers, their power state (whether sleeping or not), and their frequency. This study would like to make sure that just the right CPU capacity is made available at the right time to avoid spill-overs, while lowering the energy consumption.

This study models the overheads of all three control actions - turning on/off servers, co-locating/moving service sessions between the servers, and changing operating frequency or CPU power modes. This study considers the cost of starting/rebooting machines for bringing up servers. This study bases the cost of migrating client sessions on their memory footprints and rough estimates of network bandwidth between the server machines, and accordingly make the
<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>number of Clients</td>
</tr>
<tr>
<td>$L$</td>
<td>number of Servers</td>
</tr>
<tr>
<td>$i$</td>
<td>subscript for server $i$</td>
</tr>
<tr>
<td>$j$</td>
<td>subscript for client $j$</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>interval for DVS</td>
</tr>
<tr>
<td>$t$</td>
<td>subscript for interval for DVS</td>
</tr>
<tr>
<td>$\Delta T$</td>
<td>interval for Packing</td>
</tr>
<tr>
<td>$T$</td>
<td>subscript for interval for packing; $T = z \cdot t$, where $z$ is an integer</td>
</tr>
<tr>
<td>$\eta$</td>
<td>percentile capacity assurance</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>parameter to construct $\eta$ capacity assurance based on load distribution function</td>
</tr>
<tr>
<td>$\beta$</td>
<td>scaling parameter to account for client load correlations</td>
</tr>
<tr>
<td>$f_{i,t}$</td>
<td>frequency for server $i$ at interval $t$</td>
</tr>
<tr>
<td>$\nu$</td>
<td>normalization factor for converting frequency to capacity</td>
</tr>
<tr>
<td>$Y_{i,j,T}$</td>
<td>= 1 if client $j$ is served by server $i$ during interval $T$; 0 otherwise</td>
</tr>
<tr>
<td>$X_{i,T}$</td>
<td>= 1 if server $i$ is “on” during interval $T$; 0 otherwise</td>
</tr>
<tr>
<td>$w_{j,T}$</td>
<td>mean demand from client $j$ during interval $T$</td>
</tr>
<tr>
<td>$s_{j,T}$</td>
<td>std. dev. of demand from client $j$ during interval $T$</td>
</tr>
<tr>
<td>$W_{i,T}$</td>
<td>mean demand of packed clients on server $i$ during $T$</td>
</tr>
<tr>
<td>$S_{i,T}$</td>
<td>std. dev. of demand of packed clients on server $i$ during $T$</td>
</tr>
<tr>
<td>$WW$</td>
<td>$\sum_{j=1}^{N} w_{j}$</td>
</tr>
<tr>
<td>$SS^2$</td>
<td>$\sum_{j=1}^{N} \beta \cdot s_{j}^2$</td>
</tr>
<tr>
<td>$\overline{W}$</td>
<td>balanced Mean target for each server</td>
</tr>
<tr>
<td>$\overline{S}$</td>
<td>balanced Std. Dev. target for each server</td>
</tr>
</tbody>
</table>

Table 5.1. Notations and definitions used in this chapter

concerned server CPUs unavailable for the client sessions during the migration. Changing the frequency or the CPU power states incurs very little overheads which is negligible for the granularity (15 seconds) at which such control is implemented.

5.2 Problem formulation

This study considers a server cluster consisting of $L$ identical servers/blades, which host the workloads from $N$ types of client within an application/enterprise. These $L$ servers can the subset of whole servers’ pool, considered in chapter 4. It’s again assumed that each server is equally capable of running any type of service workload and it can host multiple types of services at the same time. The workload from a service session discussed in previous section - can only be offloaded to one server (the workload is not divisible amongst multiple servers) at any time.
However, it can be migrated between servers during the course of execution.

**Server Power Consumption Model** Each server is operated at one (may change over time) of a set of discrete CPU frequency $f$ levels between a maximum frequency $f_{\text{max}}$ and a minimum frequency $f_{\text{min}}$. Different from the power consumption model of single server in chapter 4, this chapter considers a more sophisticated power scheme for the purpose of finer frequency/voltage control. The power consumption of a server is calculated at any instant as

$$P = \begin{cases} 
0 & \text{when server is turned off}, \\
P_{\text{fixed}} + P_f \cdot f^3 & \text{CPU active/operational}, \\
P_{\text{fixed}} + P_{\text{halt}}(f) & \text{CPU in HALT, but clock is on}. 
\end{cases} \quad (5.1)$$

When the server is powered off completely, none of its components consume any power. At the other end, when the CPU is working on a client session at a frequency $f$, its dynamic power is proportional to $C \cdot V^2 \cdot f$, where $C$ is the capacitance (constant for our purposes), and $V$ is the voltage. In DVS, when the voltage is lowered, the frequency is also lowered proportionally to allow circuits to accommodate higher switching delays (caused by the lower voltage). Consequently, the dynamic power of the CPU when executing is proportional to the cubic power of the frequency.

In addition to the CPU, the server has other components such as the disk, memory, etc, which consume power independent of the CPU, and we account for those as $P_{\text{fixed}}$. This leads to the second equation in the above formula, which has also been used in related studies [33, 27].

It is also possible to only lower the voltage (without lowering the frequency), and such techniques are used only when the CPU is not operational, i.e. not executing any work, and these are usually termed as different HALT/SLEEP states [5]. The CPU is not completely shut down in these modes, rather the voltage is reduced at the same frequency, and the clock is gated to not activate all its components. The reason for doing so is because the transition time out of these HALT states is negligible compared to fully shutting down the server and bringing it back.
up again. The power consumed by the CPU in these states is still dependent on the frequency (denoted as $P_{halt}(f)$), though this value is lower than $P_f \cdot f^3$ given in the second equation. The server power will include this CPU power and the power of the other system components ($P_{fixed}$) as depicted in the third equation. Note that this study transitions the server CPU to the HALT state as soon as it finishes the work assigned to it (taking negligible time), and leave it in this state until it is given the next piece of work when it becomes operational again (taking negligible time).

![Figure 5.3. CDF of CPU Demand with $\eta$ Capacity Assurance](image)

**Service Workload and Server Capacity Constraint**  
As discussed in Section 5.1.2, memory and I/O resource needs are not the bottlenecks, and this study focuses on the CPU resources. The time-varying CPU demand of each service workload/session is expressed in terms of the server utilization when it serves the client workload at the maximum frequency. The server CPU capacity allocation is done by a combination of (i) turning on an appropriate number of servers, and (ii) operating each server at the right frequency level. One of the goals of this study is to provide a tunable knob to easily achieve trade-offs between performance and power when packing
the service workloads on the server, and this is done using a parameter $\eta$. This parameter defines
the percentile of the server CPU demanded by the clients allocated to it (pictorially shown in
Figure 5.3), which the system would like to provision. It can also be referred to as the assurance
of CPU capacity provisioning, whose motivation is given below:

- Even if the CPU demands of the clients is accurately known ahead of time for resource
  provisioning, providing 100% of these resources may not necessarily be the best option.
  Due to the highly non-linear dependence between performance and power, one may be able
to trade-off a little amount of degradation in performance for significant power savings
by providing only the $\eta$-th percentile of demand. This becomes even more important for
heavy-tailed CPU demand distributions, since a lot more resources would be needed even
for a small performance improvement beyond a point.

- The CPU demand is a random variable, and there is bound to be some uncertainty in its
  estimation of its moments. By providing only the $\eta$-th percentile value, it is quite possible
that the proposed methodology may not have to compromise on any performance at all,
while still enjoying power savings.

Let $w_{j,T}$ denote the mean CPU demand of the $j$th service workload in the $T$th time period
with duration $\Delta T$ and let $s_{j,T}$ denote the corresponding standard deviation of the CPU demand.
Then the CPU demand on server $i$ can be calculated in terms of the CPU demand of each
individual service workload offloaded to this server. If the service workloads offloaded to a server
are independent with no correlation, the mean of the sum of their CPU demands equals the sum
of their mean values and the variance of the sum equals the sum of their variances. However, when
this study analyzed the traces from the 100 clients, this study found that they are not exactly
independent. Therefore, the variance of the total CPU demand on server $i$ consists of two parts;
the first part is the sum of variances of individual client workloads offloaded to the server and
the second part is the sum of covariances of any two client workloads. For convenience, the
variance of each service’s CPU demand is scaled by a common factor, $\beta$, whose value is obtained from analyzing the traces to reflect the average correlation between different client workloads. This study then treats the scaled CPU demand from different client machines ($w_{j,T}, \beta \cdot s_{j,T}^2$) as independent workloads, and the corresponding means and scaled variances can be added.

Let $W_{i,T}$ denote the mean total CPU demand of all service workloads offloaded to server $i$ in the $T$th time period of $\Delta T$ and $S_{i,T}$ denote the corresponding standard deviation. Define $Y_{i,j,T} = 1$ to indicate that service $j$ is offloaded to server $i$ in time period $T$ and $Y_{i,j,T} = 0$ otherwise. Then $W_{i,T} = \sum_{j=1}^{N} (w_{j,T} \cdot Y_{i,j,T})$, $S_{i,T}^2 = \sum_{j=1}^{N} (\beta \cdot s_{j,T}^2 \cdot Y_{i,j,T})$. Considering an $\eta\%$ assurance for CPU provisioning, the $\eta$-percentile CPU demand on server $i$ can be represented as $W_{i,T} + \alpha \cdot S_{i,T}$, where the parameter $\alpha$ can be determined in terms of $\eta\%$ for different CPU demand distributions, e.g., if the CPU demand follows a normal distribution, $\alpha = 1.65$ for $\eta = 95\%$. Since the CPU capacity provided by server $i$ is linearly proportional to the operating frequency $f_i$, the capacity constraint on providing $\eta\%$ assurance of CPU provisioning is written as follows,

$$W_{i,T} + \alpha \cdot S_{i,T} \leq \nu \cdot f_{i,t}$$

where $\nu$ denotes the normalization parameter from frequency to CPU capacity.

**Optimization Formulation** As described earlier, since the workload is given as a sequence of server CPU resource demands at a 15 second resolution, the performance goal is to minimize the spill-over of the demand (due to failure in provisioning) from one period/interval to the next. It should be noted that the $\eta\%$ assurance for capacity provisioning has an impact on the spill-over, with a higher $\eta$ leading to lower spill-overs, with possibly higher power consumption, and vice-versa. Resource provisioning to meet these demands involves: (i) turning on/off servers, (ii) co-locating/packing the client sessions to a server, and (iii) dynamically modulating the CPU frequency (DVS). Considering the overheads involved, the server turn on/off and co-location of client sessions (which may require migration between the servers) is done at a coarser granularity...
$\Delta T$ (e.g. 1 hour), while the DVS control is done at a finer granularity $\Delta t$ (e.g. 15 seconds). Without loss of generality, this study assumes that $\Delta T = z \cdot \Delta t$ where $z$ is an integer. Let $X_{i,T} = 1$ denote server $i$ is turned on in the $T$th time period of $\Delta T$ and $X_{i,T} = 0$ otherwise.

This thesis can now formulate the constrained optimization problem of minimizing the total energy consumption subject to the constraints of $\eta \%$ capacity assurance. When doing so, note that the goal of the control algorithms should be to either keep a server completely off, or operational at a frequency $f$. Even though there is the third part of equation (5.1) to consider in power calculations, note that stretching out an execution for the entire $\Delta T$ so that the CPU is either completely off or in a DVS mode, would usually consume lower power than running at a high frequency to finish the work early and moving to a halt state. Consequently, this study does not use the third part of the equation as far as the optimization formulation is concerned, though it’s used in the actual experimental evaluation whenever the server is on but not executing.

Further, since the granularity of migration is done at $\Delta T$ (around 60 minute granularity), this study does not explicitly factor that issue in our optimization, though the overheads of migrating client sessions are included in the experiments. The constrained optimization problem can then be expressed as

$$\min \sum_{i=1}^{L} \left\{ \sum_{T} \sum_{t=1}^{z} \left( P_{\text{fixed}} + P_{f} * f_{i,t}^{3} \right) \cdot \Delta t \cdot X_{i,T} \right\}$$

subject to

- Provide at least $\eta$-percentile demand on each server
  $$W_{i,T} + \alpha \cdot S_{i,T} \leq \nu \cdot f_{i,t}; \forall i, T$$

- Mean of Total Demand for server $i$
  $$W_{i,T} = \sum_{j=1}^{N} w_{j,T} \cdot Y_{i,j,T},$$

- Var. of Total Demand for server $i$
\[ S_{i,T}^2 = \sum_{j=1}^{N} \beta \cdot s_{j,T}^2 \cdot Y_{i,j,T}; \forall i, T \]

- Client only on 1 server at any time
  \[ \sum_i Y_{i,j,T} = 1; \forall j, T \]
- No. of servers is finite
  \[ \sum_i X_{i,T} = l \leq L; \forall i, T \]

### 5.3 Solution methodologies

Since the optimization problem is a constrained nonlinear binary program, it is computationally hard, and this study uses heuristic algorithms. In particular, this optimization problem is decomposed into two decoupled subproblems. Assuming that each on-server is operating at its optimal DVS frequency, the first subproblem solves the assignment of client workloads to servers, which consists of grouping multiple client workloads together, turning on a new server if necessary, and then packing each group of client workloads to a server. It should be noted that the decision on grouping clients would not only affect the server utilization (spill-over) but also affect the power consumption significantly. Further, it may require migrating the service workloads from one server to another. The decision for this subproblem is made at the beginning of each \( \Delta T \) time period. After service workloads have been packed onto the chosen subset of on-servers, the second subproblem determines frequencies for DVS control. It uses the minimum frequency to process the workloads to minimize spill-overs. The DVS control is performed at a much smaller time granularity \( \Delta t \) and is performed independently at each server.

#### 5.3.1 Co-location/packing of client workloads

At the beginning of each \( \Delta T \) time period, this study uses the ARMA model and the Winter’s smoothing method as described in 4.3 to estimate the mean and variance of each client’s CPU demand for the upcoming \( \Delta T \) period. The required number of on-servers and the grouping/packing
For every $\Delta T$ interval

- Predict mean and variance of each client workload
- Set initial number of on-servers $l = (WW + \alpha \cdot SS)/(\nu \cdot f_{max})$
- Set $\bar{l} = \lceil F \cdot (\frac{2P_f}{P_{fixed}})^{\frac{2}{3}} \rceil$

while ($l \leq \bar{l}$ && significant reduction in energy consumption){
  - Sort all client workloads by CPU demand in decreasing order
  - Set balanced CPU demand $W + \alpha S$
    - Set balanced-mean $\bar{W}$ and balanced std. dev. $\bar{S}$
    - First-Fit/Next-Fit bin packing (check 1-D/2-D capacity constraints)
    - Set $f_i = (W_i + \alpha \cdot S_i)/\nu$
  - Calculate the energy consumption
  - Set $l = \text{number of bins from feasible bin packing result}$
    - if it is larger than the $l$ before bin packing
    - otherwise increase $l$ by one
}

**Figure 5.4.** Pseudo-code for Packing

of client sessions to the on-servers are determined using these estimates. The co-location/packing problem is reformulated as a bin packing problem with the pseudo-code listed in Figure 5.4 which is explained below.

Since the power consumption is a cubic function of the frequency, the case where all given servers operate at the same frequency consumes the lowest power compared to cases where servers run at different frequencies in order to provide the same aggregate CPU capacity as the former case. By assuming that each server operates at this optimal DVS frequency (the minimum feasible frequency to provide the CPU demand) in this packing problem, this study can approximate the average frequency of a server for the $\Delta T$ time period by the CPU demand on this server as $(W_i + \alpha \cdot S_i)/\nu$. Since frequency directly relates to the capacity on each server, the co-location/packing of workloads which provisions the same CPU demand $W_i + \alpha \cdot S_i$ across all servers (all running at the same frequency) consumes the lowest energy. Alternatively, if this study considers the mean and variance as two separate metrics for workloads, it can be shown that the packing which gives a balanced-mean and balanced-variance of CPU demand across a given number of servers has the lowest energy consumption. Note that this result holds at the data domain of the interest of this study with the assumption relaxing the indivisibility of
workloads, and separability of mean and standard deviation of separate resources.

Consequently, the minimization of energy consumption is reduced to a bin packing problem with 1 or 2 dimensional constraint checking of getting to a balanced-mean target and a balanced-variance target of the CPU demand across the servers.

Let $WW$ denote the mean of the aggregate demand from all $N$ clients, i.e., $WW = \sum_{j=1}^{N} w_j$. Let $SS^2$ denote the corresponding (scaled) variance, i.e., $SS^2 = \sum_{j=1}^{N} \beta \cdot s_j^2$. For a given $l$ number of on-servers, the balanced-mean target of the client workload for each server is $\bar{W} = WW/l$ and its balanced standard deviation target is $\bar{S} = SS^2/l$. Thus the balanced $\eta$ percentile CPU demand is $\bar{W} + \alpha \cdot \bar{S}$. The proposed algorithm finds the number of on-servers $l$ that gives the lowest energy consumption by iteration. The lower bound of $l$ is given by the ratio of the aggregate CPU demand from all clients ($WW + \alpha \cdot SS$) divided by $\nu \cdot f_{max}$, assuming that each workload is divisible among multiple servers (though this is not really true in practice). The upper bound $\bar{l}$ for the number of on-servers is determined based on the observation that one needs to balance between the number of on-servers and the operating frequency of each server. Essentially, there is a break-even number $l^*$ for on-servers, beyond which the increase in energy consumption due to $P_{fixed}$ by turning on one more server exceeds the energy reduction due to possible lower operating frequency for each already on-server. Define an aggregate frequency $F$ needed to meet the aggregate CPU demand ($WW + \alpha \cdot SS$) from all client workloads, i.e. $F = (WW + \alpha \cdot SS) / \nu$. If each client workload is considered divisible across multiple servers, the average frequency of each server in $\Delta T$ time interval is $F/l$. Then the total energy consumption is approximated as $E = l \cdot (P_{fixed} + P_f \cdot (F/l)^3) \cdot \Delta T$. The break-even number $l^*$ for on-servers is calculated by setting the first derivative of $E$ with respect to $l$ to zero, i.e., $l^* = [F \cdot (2P_f / P_{fixed})^{\frac{1}{3}}]$. This break-even number $l^*$ becomes an upper bound for the number of on-servers ($\bar{l}$) if the workload is not divisible.

This study describes the implementation of the bin packing for each value of $l$ as follows:

- **Searching the Bins:** This study first sort the client workloads in decreasing order of CPU
demand \((w_j + \alpha \cdot s_j)\). Then a Next-Fit or First-Fit algorithm is used to place the client workload on servers. In Next-Fit, it tries to pack the client into the currently active bin (server). If it cannot fit - the criterion for determining whether it can fit is explained in the next paragraph - in this bin, then it closes the bin and moves on to the next (the algorithms never return back to closed bins). In First-Fit, it always searches the bins from the oldest (to newest), to see whether it can fit. A new bin is opened only when none of the older bins can accommodate this client.

- **Criteria for Fit:** Determining whether a client can fit in a bin, while balancing the load across the servers, can be expressed as either a one-dimensional or a two-dimensional capacity constraint. In the one-dimensional constraint, each server’s capacity (i.e. the demand of all clients allocated to that server) target is calculated as \(\bar{W} + \alpha \cdot \bar{S}\), and the criteria for whether a client can be put in this bin \(i\) depends on whether \(W_i + \alpha \cdot S_i\) exceeds this capacity when the new client is added. \(\bar{W}\) and \(\bar{S}\) can also be used as two dimensions separately in constraint checking (which this study calls the two-dimensional capacity constraint), where the server’s capacity target is estimated as \((\bar{W}, \alpha \cdot \bar{S})\), and the proposed approach wants to ensure that each of the two dimensional values when the new client is added does not exceed this capacity. Both these criteria try to balance the distribution of load across the servers (which it’s mentioned above to be the goal), though the two-dimensional constraint can be a more stringent specification than the first.

After the bin packing, the proposed metrology sets the frequency of each server \(f_i\) based on the CPU demand from all workloads packed to it, \((W_i + \alpha S_i)/\nu\) and then calculate the resulting energy consumption. The proposed approach then iterates \(l\) between the lower and upper bounds, until it gets to an \(l\) value which does not significantly reduce the energy consumption any more.
5.3.2 Algorithm for proactive DVS

One of the complications this study faces in frequency setting for each time interval $\Delta t$ is that in addition to predicting the client workload for that interval, the proposed approach also needs to compensate for the spill-over of incomplete work from the previous interval (or even under-utilization in the previous interval). These two parts are explained below, and the pseudo-code of our algorithm is given in Figure 5.5.

**Predicting client demand:** Traditionally prior time windows are used to estimate the load for the next window, and a fixed number of such windows have been used in several prior studies for frequency setting [96, 65, 42, 40]. From workloads, it’s found that a varying number of windows, History Window Size ($HWS$), is a better option due to time-varying behavior. To find the $HWS$ to use for the next $\Delta t$, the proposed DVS first determines the maximum significant lags ($I$) of the time series of the CPU demands. It then sees how many windows ($HWS$) in the past (upto $I$) that it needs to look at to get the lowest average cumulative values for the square error between the prediction and the actual value. Note that even though the proposed DVS algorithm would like to use the actual value of demand in calculating these errors, that information is not even available to the control algorithm because of spill-overs and the actual utilization can only be monitored. This utilization ($U$) is a function of both the new incoming load, and also the spill-over from the previous interval(s). However, it’s found that using the utilization, instead of the actual load still served purpose of this study.

**Setting DVS frequency:** As noted above, the proposed DVS scheme needs to accommodate the new incoming demand, and compensate for spill-overs/under-predictions and under-utilization/over-predictions from the prior period(s). Basing the setting only on the load prediction would not compensate for these errors in prior periods. Consequently, the proposed speed setting adjusts the prediction ($U_{i,t}$) of load by increasing or decreasing it based on whether the
For every t interval and i server
/* Predict workload on server i */
Set History Window Size (HWS)
Predict Utilization $U_{i,t}$ from HWS past windows
/* Set Operating frequency */
If under-predicted in prev. interval, then
$$U_{i,t} = U_{i,t} + \gamma \cdot D_{i,t}$$
Else if over-predicted in prev. interval then
$$U_{i,t} = U_{i,t} - \gamma \cdot D_{i,t}$$
Set $f_{i,t} = \lceil U_{i,t}/\nu \rceil$

Figure 5.5. Pseudo-code for DVS Algorithm

previous interval had a spill-over or was under-utilized. This compensation is dependent on how
much the difference in utilization between successive intervals is averaging over its last HWS
intervals (expressed as $D_{i,t}$ in Figure 5.5). Note that $\gamma$ is a parameter the proposed DVS can
tune for the mispredictions. Once it predicts the utilization for the next window, it can use the
normalization factor $\nu$ to get the corresponding frequency setting.

5.4 Experimental Evaluation and Results

5.4.1 Experimental setup

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Clients</td>
<td>100</td>
</tr>
<tr>
<td>Max. servers</td>
<td>45</td>
</tr>
<tr>
<td>$\Delta T$</td>
<td>60 minutes</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>15 seconds</td>
</tr>
<tr>
<td>Total Duration</td>
<td>48 hours</td>
</tr>
<tr>
<td>$\eta$</td>
<td>50, 85, 95-th percentiles</td>
</tr>
<tr>
<td>Mig. Bw/Server</td>
<td>55 MBytes/sec</td>
</tr>
<tr>
<td>Frequency (GHz)</td>
<td>1, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4</td>
</tr>
<tr>
<td>Active Power (W)</td>
<td>56, 59, 61.5, 64, 67.5, 75.5, 88, 101</td>
</tr>
<tr>
<td>Halt Power (W)</td>
<td>50, 50.65, 51.3, 51.9, 52.5, 54.5, 57.5, 63</td>
</tr>
</tbody>
</table>

Table 5.2. Simulation parameters

As explained earlier, our client workloads are obtained from real user sessions of a Fortune
500 Financial institution during a two-week period in 2004. A hundred client traces, each of
which is 48 hours long, have been picked for our simulation and evaluation. This study allows
upto $L = 45$ servers for all the schemes. The set of frequencies for each server ranges from 1GHz to 2.4GHz with 0.2 GHz increment, denoted as ($f_1 \ldots f_8$).

The corresponding power consumption values for different frequencies, listed in Table A.1, are values from a server with an AMD processor and 4GB RAM which include the $P_{\text{fixed}}$ component. The server bandwidth for migration of each client session is calculated using current network bandwidths, and the sharing of the single Gigabit link into the server for both sending and receiving. Note that the actual cost for migration is based on the client session memory footprint available in the logs. Three $\eta$ values, 50, 85 and 95 percentiles are used and the corresponding $\alpha$ values from our empirical CDF demand data are 0, 0.9 and 1.6 respectively.

The metrics of concern are the energy consumption (given in kilowatt-hours) and the spill-overs (given as the percentage of windows of granularity 15 seconds incurring these spill-overs), both of which need to be minimized. The energy consumption is cumulated across the servers and presented for the 48 hour duration, while the spill-overs are averaged across all the servers.

<table>
<thead>
<tr>
<th># servers</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>fi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fi</td>
<td>38.2</td>
<td>53.2</td>
<td>67.3</td>
<td>80.6</td>
<td>93.0</td>
<td>104.5</td>
<td>114.6</td>
<td>124.4</td>
</tr>
<tr>
<td>fi</td>
<td>42.6</td>
<td>58.9</td>
<td>74.2</td>
<td>88.8</td>
<td>102.2</td>
<td>114.9</td>
<td>126.1</td>
<td>136.6</td>
</tr>
<tr>
<td>Optimal</td>
<td>38.6</td>
<td>51.1</td>
<td>63.2</td>
<td>74.6</td>
<td>85.4</td>
<td>95.2</td>
<td>104.4</td>
<td>112.4</td>
</tr>
<tr>
<td>Pro-active</td>
<td>39.2</td>
<td>51.9</td>
<td>64.0</td>
<td>75.4</td>
<td>86.3</td>
<td>95.8</td>
<td>108.0</td>
<td>112.2</td>
</tr>
<tr>
<td>IBM-ARL</td>
<td>39.0</td>
<td>51.7</td>
<td>63.8</td>
<td>75.3</td>
<td>85.9</td>
<td>95.8</td>
<td>105.1</td>
<td>113.3</td>
</tr>
<tr>
<td>OnDemand</td>
<td>39.9</td>
<td>53.3</td>
<td>65.9</td>
<td>77.7</td>
<td>88.7</td>
<td>98.8</td>
<td>107.9</td>
<td>116.3</td>
</tr>
</tbody>
</table>

(a) Energy consumption (KWh)

<table>
<thead>
<tr>
<th># servers</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>fi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fi</td>
<td>48.59%</td>
<td>28.34%</td>
<td>19.07%</td>
<td>14.17%</td>
<td>11.21%</td>
<td>9.29%</td>
<td>8.06%</td>
<td>7.10%</td>
</tr>
<tr>
<td>fi</td>
<td>41.63%</td>
<td>23.20%</td>
<td>15.36%</td>
<td>11.34%</td>
<td>8.85%</td>
<td>7.30%</td>
<td>6.14%</td>
<td>5.38%</td>
</tr>
<tr>
<td>Optimal</td>
<td>41.46%</td>
<td>23.25%</td>
<td>15.43%</td>
<td>11.18%</td>
<td>8.88%</td>
<td>7.23%</td>
<td>6.17%</td>
<td>5.42%</td>
</tr>
<tr>
<td>Pro-active</td>
<td>46.79%</td>
<td>28.65%</td>
<td>20.45%</td>
<td>16.98%</td>
<td>14.05%</td>
<td>11.56%</td>
<td>10.31%</td>
<td>10.71%</td>
</tr>
<tr>
<td>IBM-ARL</td>
<td>54.22%</td>
<td>39.89%</td>
<td>32.39%</td>
<td>27.29%</td>
<td>23.98%</td>
<td>21.59%</td>
<td>19.60%</td>
<td>18.17%</td>
</tr>
<tr>
<td>OnDemand</td>
<td>47.00%</td>
<td>29.81%</td>
<td>21.97%</td>
<td>17.37%</td>
<td>14.60%</td>
<td>12.63%</td>
<td>11.18%</td>
<td>10.16%</td>
</tr>
</tbody>
</table>

(b) % Spill-over

Table 5.3. Naive Packing for client sessions
5.4.2 Results

For the first step, this study compares the proposed packing schemes with a rather \textit{naive} approach, where this study randomly allocate the clients to the given number of servers, with each client equally likely to go to any server. Consequently, in the naive scheme, the packing of servers is done by purely balancing the number of client sessions across the servers, oblivious to the load imposed by each session. Please note that the naive scheme is to form independent queues with respect to each server, and requests within each queue are served in FCFS fashion. On the other hand, the scheduling policy discussed in chapter 4 is to dispatch arriving requests to the first available server. Since this balancing scheme can be highly sensitive to the number of servers, this study runs experiments with different number of servers ranging from 10 to 45. Each experiment has been averaged over 100 repetitions with different random number seeds.

For the proposed packing solution, note that four possibilities have been identified in section 5.3.1 - NF-1D, FF-1D, NF-2D and FF-2D - which is based on the (i) technique used for client co-location (Next-Fit or First-Fit), and (ii) the constraint for checking server capacity which can be 1-Dimensional ($\bar{W} + \alpha \bar{S}$), or 2-Dimensional ($\bar{W}, \alpha \cdot \bar{S}$).

For the second step, this study considers four possibilities for frequency control with DVS - Optimal, Pro-active, IBM-ARL, and OnDemand. The Optimal algorithm assumes the workload for the next interval is known a priori, and can perform perfect frequency setting. The Pro-active approach, which uses prediction of load for the next interval, is the proposal described in section 5.3.2. The IBM-ARL mechanism is a reactive one, that has been proposed earlier in [33]. The OnDemand algorithm is also a reactive solution used in Linux for frequency setting [61], which is traditionally more conservative. For reference, this study also includes experiments where the frequency setting is not modified at all during the entire experiment, and results are presented for the highest two frequency settings ($f_8 = 2.4$ GHz and $f_7 = 2.2$ GHz). Note that in all these DVS algorithms, in addition to any frequency control, the \textit{halt} states are used in the respective
frequencies whenever the CPU does not have any work to perform.

Tables 5.3 (a) and (b) show the energy consumption and %spill-overs for the naive packing schemes with different number of servers and different schemes for frequency control. Table 5.4 presents the corresponding information for our schemes with different capacity assurance values ($\eta$).

### 5.4.2.1 Comparing packing mechanisms

<table>
<thead>
<tr>
<th>Packing</th>
<th>DVS</th>
<th>Egy</th>
<th>Perf.</th>
<th># svr</th>
<th>Egy</th>
<th>Perf.</th>
<th># svr</th>
<th>Energy</th>
<th>Perf.</th>
<th># svr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alg.</td>
<td>Alg.</td>
<td>KWh</td>
<td>spill-overs per hr</td>
<td>Alg.</td>
<td>Alg.</td>
<td>KWh</td>
<td>spill-overs per hr</td>
<td>Alg.</td>
<td>Alg.</td>
<td>KWh</td>
</tr>
<tr>
<td>NF-1D</td>
<td>Optimal</td>
<td>42.77</td>
<td>44.96%</td>
<td>11.4</td>
<td>44.75</td>
<td>42.83%</td>
<td>10.8</td>
<td>51.15</td>
<td>30.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-1D</td>
<td>Optimal</td>
<td>42.77</td>
<td>44.96%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-2D</td>
<td>Optimal</td>
<td>42.77</td>
<td>44.96%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>57.80</td>
<td>13.61%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-2D</td>
<td>Optimal</td>
<td>42.77</td>
<td>44.96%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>57.80</td>
<td>13.61%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-1D</td>
<td>Pro-active</td>
<td>43.68</td>
<td>42.12%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-1D</td>
<td>Pro-active</td>
<td>42.82</td>
<td>43.67%</td>
<td>10.8</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-2D</td>
<td>Pro-active</td>
<td>43.68</td>
<td>42.12%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-2D</td>
<td>Pro-active</td>
<td>42.82</td>
<td>43.67%</td>
<td>10.8</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-1D</td>
<td>IBM-ARL</td>
<td>43.52</td>
<td>47.73%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-1D</td>
<td>IBM-ARL</td>
<td>42.61</td>
<td>48.36%</td>
<td>10.8</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-2D</td>
<td>IBM-ARL</td>
<td>43.52</td>
<td>47.73%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-2D</td>
<td>IBM-ARL</td>
<td>42.61</td>
<td>48.36%</td>
<td>10.8</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-1D</td>
<td>OnDemand</td>
<td>44.18</td>
<td>43.96%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-1D</td>
<td>OnDemand</td>
<td>43.04</td>
<td>45.06%</td>
<td>10.8</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-2D</td>
<td>OnDemand</td>
<td>44.18</td>
<td>43.96%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-2D</td>
<td>OnDemand</td>
<td>43.04</td>
<td>45.06%</td>
<td>10.8</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-1D</td>
<td>f7</td>
<td>41.96</td>
<td>68.49%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-1D</td>
<td>f7</td>
<td>39.93</td>
<td>70.50%</td>
<td>10.8</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-2D</td>
<td>f7</td>
<td>41.96</td>
<td>68.49%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-2D</td>
<td>f7</td>
<td>39.93</td>
<td>70.50%</td>
<td>10.8</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-1D</td>
<td>f8</td>
<td>45.22</td>
<td>39.96%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-1D</td>
<td>f8</td>
<td>44.52</td>
<td>40.90%</td>
<td>10.8</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>NF-2D</td>
<td>f8</td>
<td>45.22</td>
<td>39.96%</td>
<td>11.4</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
<tr>
<td>FF-2D</td>
<td>f8</td>
<td>44.52</td>
<td>40.90%</td>
<td>10.8</td>
<td>42.83</td>
<td>42.83%</td>
<td>10.8</td>
<td>54.34</td>
<td>20.42%</td>
<td>10.8</td>
</tr>
</tbody>
</table>

Table 5.4. Comparing the schemes for Packing Algorithm and DVS Schemes.

This study notes the following observations about packing mechanisms from experimental results:

- The results for the naive scheme (Table 5.3) suggest that it is difficult to optimize both performance and energy goals simultaneously if smarter packing of service workloads is not performed on the servers. Better performance comes at the expense of higher energy consumption and vice-versa. For instance, achieving less than 15% spill-overs even with an optimal DVS algorithm for the naive packing, takes over 30% more energy than our
schemes (using an $\eta=85\%$). With the naive packing algorithm, it needs a larger number of servers (20 or higher) to get acceptable performance (say less than 15% spill-overs), than the number of servers used on the average (around 15) with our mechanisms (compare to Table 5.4). Consequently, the energy savings with a better packing algorithm compared to the naive approach will be more significant with more stringent performance demands.

- For the same performance requirements, note that the gap between the energy consumption of a DVS based scheme and the one always running at the maximum frequency ($f_8$) is higher for the naive scheme compared to the same gap in our solutions. In fact, this gap grows when moving from 25 to 45 servers in the naive scheme suggesting that DVS is more important when (i) the packing algorithm is not as effective, and (ii) resource is over-provisioned.

- Moving on to the proposed packing strategies, this study sees evidence of the advantage of using a tunable capacity assurance parameter ($\eta$), which can help attain good design points in the performance-energy space. For instance, in the Optimal DVS execution of NF-1D, going from 50% to 85% capacity assurance, reduces the spill-overs by over 60%, while the energy consumption goes up around 12%. However, further reductions in spill-over by going to an $\eta$ value of 95%, can result in over 30% increase in energy consumption, i.e. the marginal costs increase. One could use this parameter, to avoid accommodating heavy-tailed CPU demand distributions where one may pay a much higher energy cost to get marginal performance improvements.

- When the choice of bin selection are compared- Next-Fit versus First-Fit - the former is more conservative and over-provisions capacity. This is because the proposed scheme closes the bins and does not come back to them later, while First-Fit can pack the bins to lessen the wastage. Consequently, Next-Fit does slightly better in terms of spill-overs but does slightly worse in terms of energy consumption. The conservative behavior worsens as higher levels of CPU assurance ($\eta$) is chosen.
- Similar to the previous observation, a two-dimensional server capacity constraint is more conservative than a linearized one-dimensional constraint, making the former choice slightly better for spill-overs and the latter slightly better for energy consumption. As before, this conservative behavior worsens with higher CPU assurance.

- The above two observations indicate that the proposed approach could trade-off the aggressiveness of the bin selection algorithm and the dimensionality of the server capacity constraint against each other by opting for NF-1D and FF-2D, which fall in-between the other two in terms of both spill-overs and energy.

![Figure 5.6. Spill-over for Hour 47 with Optimal DVS](image)

- In order to show that the proposed schemes are indeed trying to balance the load across the servers (which this study pointed out to be a goal when packing in section 5.3.1 for a given CPU demand), this study focuses on the 47th hour of the execution. Note that after applying a packing algorithm, each server has a mean and standard deviation of the load for this interval, and the goal of this study was to balance each of them. If the coefficient of variation (COV) of this mean and standard deviation across different servers are examined, the corresponding values are 0.4 and 0.8 for NF-1D, 0.5 and 0.6 for FF-2D, and 0.7 and
0.8 for the naive scheme. The naive scheme gives relatively less spatial load balance and its effect is clearly visible in Figure 5.6 where the spill-overs for each scheme are shown for each server in this interval. Between NF-1D and FF-2D, though the spatial variance of the mean are roughly the same (i.e. the mean load is roughly the same across the servers), the standard deviation of the temporally varying load on a server is more balanced across the servers in FF-2D - because of using a 2-dimensional constraint - compared to NF-1D. Consequently, FF-2D has slightly lower (and balanced) spill-overs across the servers compared to NF-1D. This is also important from a fairness point of view since a client session should not be penalized for going to a specific server.

5.4.2.2 Comparing DVS mechanisms

The following observations regarding the DVS mechanisms are made:

- As noted earlier, the results for the naive packing scheme (Table 5.3), suggest that when one needs to keep spill-overs within stringent bounds, power management with DVS is equally important since this scheme tends to require a lot more servers to achieve comparative performance as the smarter packing schemes.

- This study also notices somewhat similar trends about the importance of DVS with the smarter packing schemes as well, though to a lesser extent. When the level of capacity assurance is lower, the proposed schemes are able to employ a smaller number of servers (see the values for $\eta$ of 50% and 95%, with the former requiring between 30-50% lower number of servers on the average). At the smaller assurance values, the difference between Optimal and $f8$ is only around 5-10%, making the importance of DVS less significant. However, with higher assurance requiring more number of servers, the role of DVS contributes to around 20% difference in energy consumption.

- This study also notes that the energy consumption of Optimal with 85% capacity assurance
is not very different from that for 50% assurance running at highest frequency \( f_8 \). This means that with a good DVS algorithm, the proposed approach would enjoy the performance benefits of larger resource provisioning while really not compromising on the energy consumption.

- With the energy benefits due to DVS limited by the gap between energy consumption of the \( f_8 \) and Optimal executions, there is not a significant difference between the practical DVS implementations - Pro-active, IBM-ARL and OnDemand at smaller \( \eta \) values. Figure 5.7 (a) shows the fraction of time and the fraction of energy spent in each frequency state for the FF-2D execution with an \( \eta \) of 50%. As it can be seen, the profile for the three practical DVS algorithms are not very different from that for the Optimal. Further, the difference between \( f_8 \) and Optimal is itself not significant because, at these small capacity assurances, the resources are rather heavily utilized. Consequently, the frequencies are pushed more towards the higher side even in the Optimal DVS algorithm. Note that \( f_8 \) does use the halt state whenever the servers are on but idle. At higher \( \eta \) values, the growing gap between Optimal and \( f_8 \) makes DVS quite important, bringing out the differences between the practical schemes as illustrated by the differing profiles in terms of time and energy in Figure 5.7 (b) for \( \eta = 95\% \).

- The differences in DVS algorithms are more notable on the performance consequences of the execution. This study sees that the reactive IBM-ARL algorithm gives slightly lower energy consumption than the more conservative Linux OnDemand algorithm, but at the expense of much higher spill-overs. The Linux algorithm is more conservative in provisioning, and spends more time in higher frequencies than IBM-ARL as can be seen in the profile graphs for \( \eta = 95\% \). Our Pro-active scheme, on the other hand, gives lower energy consumption with spill-overs comparable to the OnDemand algorithm. This is because the OnDemand algorithm finishes the work faster and spends more time idling (Halt), rather than stretching
Figure 5.7. DVS Profile of Time and Energy

out the execution. Our Pro-active algorithm has a lower idle time than the Linux algorithm.
5.5 Concluding remarks

One of the important considerations when consolidating client sessions on back-end servers for applications is the necessity to provision the right amount of resources and to effectively utilize these resources towards meeting the performance goals while minimizing energy consumption. This study has presented the first formulation of this goal as a constrained optimization problem and proposed a two step solution for server resource provisioning, client session co-location on these servers, and power management. In the first step, this study poses the packing of services to servers as a bin-packing problem with the constraint of balancing the distribution (the mean and standard deviation specifically) of load across the servers, which can be shown to give the lowest energy consumption for a given capacity demand. This study presents different algorithms for packing these servers and checking server capacity to achieve this balance. The second step performs DVS control to adjust the frequency/voltage of execution for power savings, and this thesis used a prediction-based, pro-active technique for this step. The cost of running our optimization algorithm for packing is only a few seconds, and this cost is negligible compared to the time granularity (60 minutes) of control. The DVS algorithm is completely distributed with each server making frequency choices independently at a 15-second resolution.

This study evaluated these alternatives with 100 real client sessions from a large financial institution. If performance is not very important, then one can provision low server capacity, making energy management and DVS less important. However, when one needs to provision enough server capacity to ensure responsiveness to client sessions, both the issues of (i) allocating this capacity to the clients, and (ii) controlling the energy consumption of the servers, become very important. This thesis showed how proposed smarter co-location schemes are able to achieve more stringent performance requirements with a lower number of servers than a naive packing strategy. Further, the proposed Pro-active DVS algorithm can provide better energy savings than the more conservative Linux OnDemand algorithm, and better performance than the previously
proposed algorithm from IBM. More importantly, the proposed gmechanism allows a tunable set of knobs - the capacity assurance parameter, choice of server selection and constraint checking for bin-packing - which allows one to attain different points of interest in the performance-power design space. It is possible to attain operating points which trade-off a small performance loss for significant reduction in energy consumption, and vice-versa, with these knobs.
Server Level Optimization

Power-Performance management can be achieved by frequency modulation of servers, as a complimentary methodology to system and application level solutions. Frequency control of servers can be executed by either central controller of each application (as considered in chapter 4) or local controller residing at each server, depending on the geographical distribution of the entire system. However, as mentioned in chapter 5, the effectiveness of centralized DVS policy is enhanced in presence of a load-balancer. A central controller requires the presence of a robust and scalable monitoring system and availability of huge network bandwidth. However, servers allocated to a application in a web hosting center may spread out geographically and have their own queues and buffers. Under such scenarios, often the centralized frequency controller imposes huge load on network by sending monitored data and control-decisions quite frequently and interfere with the overall system performances. To address systems without capacity for centralized DVS scheme, this chapter proposes an optimal frequency control at an individual server level.

Most of the existing DVS mechanisms for web hosting centers and mobile devises operate in a reactive mode. This type of DVS policies generally work based on different performance system
parameters, such as processor utilization, and other system parameters [61, 33]. These policies do not take into consideration the measures of future workloads while adjusting frequencies. Another type of policies is proactive in nature, which performs estimation of future workload, without sophisticated workload model; DVS methodology devised in chapter 5 is an example of such policy. However, a lot of data-processing required for estimation makes the estimation process cumbersome and even then, inaccuracies present in estimated data often affects the performance of these policies. Chapter 5 shows that the performance of such policies still leave much to be desired when compared to the performance of optimal policy (which has exact knowledge of incoming workload). Hence, performance of DVS policy can be improved upon with help of more detailed workload-modeling (such as, fluid modeling) as proposed by this chapter. With the presence of such detailed workload modeling, this chapter employs Markov Decision Process (MDP) to further facilitate the process of decision-making.

This study applies MDP on designing the frequency controller at a single server at the web hosting center. A MDP models the state of the system as stochastic process and takes decision accordingly. It needs to consider the states of the system, actions, transition probabilities of possible states under all actions, and reward function over decision epochs. In order to obtain optimal decision rules, MDP requires intensive computation to calculate transition probabilities and run solving algorithms, such as policy iteration. Optimal decision rules, obtained from it, specify the optimal action that should be taken at each state. In other words, the main computation overhead of MDP only lies at the beginning of planning horizon and only negligible overhead is added during on-line execution to the system. The frequency controller needs to monitor the state of the system closely at every decision epoch and executes frequency modulation in DVS following optimal rules. For the implementation viewpoint, the state of system is suggested to be defined by available and relevant information demanding very little monitoring and computation, such as remaining workloads in the system.

The main difficulty of constructing MDP for frequency control lies on deriving the transition
probability from underlying traffic. As pointed at the beginning of this thesis, traffic of web
hosting centers is usually self-similar, whose performance analysis are proposed by several early
studies [22, 94, 93, 92, 19] and has been shown as a very complicated task. Moreover, such type
of traffic is highly varying and takes much long time to stabilize [73]. Transient analysis of such
workload is in urgent need, especially for a refined granularity design of DVS. This adds another
dimension to traffic modeling combined to MDP. Often, the requests under such type traffic flow
into the system back to back, and it therefore seems like the fluid. This motivates this chapter to
address these issues by adopting stochastic fluid modeling, which has been shown as one of the
successful models for self-similar like traffic. Transient analysis of fluid model can be obtained
by some trackable solutions [71]. It’s further utilized as information for transition probability in
MDP to derive the optimal decision rules.

The specific contributions made by this chapter are the following: (a) it’s the first MDP formu-
lation based on fluid model for frequency control/DVS, with the objective of energy minimization
and constraints on system performance. Two types of fluid modeling, CTMC and DTMC, are
applied to obtain transition probabilities. Both models only demand information that requires
negligible computational overhead during online execution. (b) Based on the MDP formulation,
this study derives the structure of frequency control policy, where optimal frequency at decision
epoch, increase with the remaining workload in the system. Such control policies (derived from
both CTMC and DTMC) are well validated to maintain system performance through synthetic
traces and actual web traces, compared to other existing frequency control algorithms.

6.1 System model and problem formulation

This study considers a single server and single queue system (dedicated to a specific application)
to deliver types of services assigned by scheduler at the application level. Each request requires
files, which requires certain service capacity from the server to adhere to SLA. Within an opera-
tional server, a frequency controller has limited choices of discrete frequency levels. Verification in section 4 of chapter 4 shows that operating frequency can be assumed to be linearly related with service/computing capacity. Power consumption is an increasing function of frequency at each level. The exact computation of power consumption corresponding to each frequency considered in this chapter follows the equation 5.1. Whenever there is no pending work at the system, the server automatically switches to HALT/SLEEP state that consumes less power and incurs negligible overheads (in terms of transition-time and transition-energy), compared to shutting down the server.

In order to minimize energy consumption at such a server during certain planning horizon, the frequency controller can set required frequency at every decision epoch to execute the request while adhering to SLA. A frequently used metric within SLA is average response-time, as discussed in chapter 4. Response time depends on pending workloads within the server. Therefore,
bounding the remaining workloads below certain threshold can deliver the required performance. The constraints on performance and objective for energy are mathematically presented in 6.1, with assumption that there are $N$ decision epochs of length of $\tau$ each during the planning horizon.

Operating within a finite range of discrete frequency levels on each server is one way to maintain performance and save on power consumption. The other two available control variables for a single server power management is servers on-off and transition to HALT/SLEEP state as shown in equation 5.1. Since servers on-off is coordinatively conducted at system and application level, the decentralized frequency controller considered in this chapter can’t execute the option of server on-off, which requires centralized information. Without considering servers on-off, when the whole sequence of incoming workloads is known for every DVS control point, this study (in chapter 5) shows that operating at minimum frequency to satisfy SLA is the optimal policy, whenever there’s workload in the system. Switching to HALT state (when idle) is one available energy-saving technique, but will be only executed when there is no pending work. However, in reality, future workloads information can only be obtained by estimation. Designing DVS to incorporate the effect of HALT state needs to be coupled with estimation of futures workloads. The complexity and intractability of such a design increase by a great extent, compared to DVS design with only those discrete levels of frequencies. Therefore, the proposed methodology in the following section only adopts the second term in equation 5.1, that does not involve HALT state. Nevertheless, this chapter evaluates the effect of HALT state in experiment section.

The system under consideration allows the frequency controller to adjust the frequency periodically with fixed time epoch, $\tau$. The length of the epoch is decided by the available technology as well as the intervals for resource allocation decisions made at system and application level. The length of such an epoch, $\tau$, is comparatively smaller than intervals of server allocation and server-application mapping. In principle, it is relatively longer stationary time window, compared to the length of frequency control, $\tau$. This chapter essentially focuses on minimizing total power consumption over such finite $N$ decision epochs (each of length $\tau$) within such a station-
<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>fluid level</td>
</tr>
<tr>
<td>$z$</td>
<td>state of fluid source, On or OFF</td>
</tr>
<tr>
<td>$f$</td>
<td>available frequency $f_1, f_2, \ldots f_8$</td>
</tr>
<tr>
<td>$1/\alpha$</td>
<td>mean ON time</td>
</tr>
<tr>
<td>$1/\beta$</td>
<td>mean OFF time</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>fluid generating rate</td>
</tr>
<tr>
<td>$M$</td>
<td>generating matrix of CTMC ON-OFF model</td>
</tr>
<tr>
<td>$Q$</td>
<td>generating matrix of CTMC ON-OFF model</td>
</tr>
<tr>
<td>$D$</td>
<td>drift matrix</td>
</tr>
<tr>
<td>$H$</td>
<td>probability of $x(t) &lt; x$ under certain ON-OFF states</td>
</tr>
<tr>
<td>$c$</td>
<td>energy coefficient of frequency</td>
</tr>
<tr>
<td>$w$</td>
<td>response time</td>
</tr>
<tr>
<td>$t$</td>
<td>subscript for decision epoch for frequency control</td>
</tr>
<tr>
<td>$\tau$</td>
<td>length of decision epoch</td>
</tr>
<tr>
<td>$r$</td>
<td>reward function in MDP</td>
</tr>
<tr>
<td>$V$</td>
<td>optimality equation in MDP</td>
</tr>
</tbody>
</table>

Table 6.1. Notations and definitions in chapter 6

ary interval. In addition to power consumption, the performance constraint needs to be met at decision epoch need. The minimum frequency satisfying this constraint for each $\tau$ also consume minimum energy. However, frequency needs to be set at the beginning of each interval, where the incoming workloads within $\tau$ is unknown. The notations used in the equation below have been defined in 5.

$$
\min \sum_{t=1}^{N} (P_{\text{fixed}} + P_f \cdot f^3) \cdot \tau \\
E(w_t, f) \leq \bar{W} \\
f \in \{f_1 \prec f_2 \cdots \prec f_8\}
$$

6.2 Solution methodologies

In order to design a proactive frequency control scheme for a refined granularity, this study proposes to use Markov Decision Processes (MDP) to construct the optimal frequency decision
rules, depending only on the state of the stochastic system. Since the information of the remaining workloads, \( x \), is readily available by low monitoring overhead, it is utilized to represent the current state of the system. Thus, the complication of estimating the future can be avoided easily. As mentioned earlier, the high and varying volumes of workloads from web hosting-centers show characteristics of self-similar or self-similar like traffic, whose performance measures can not be accurately obtained by simple Poisson processes of arriving requests. The first task of constructing a MDP for web servers’ DVS is to build a transition probability that is able to characterize the transition of underlying stochastic web traffic loads.

The remaining workloads at each server under self-similar traffic are highly varying. The system with such traffic takes long time to stabilize under certain system parameters, such as operating frequency. Therefore, the transient analysis becomes ever more critical in a refined control granularity, in which frequency might be changed quite often. For the workload considered in this study, the arrival-rate is very high and hence, the sequence of incoming requests can be efficiently approximated as a continuous flow of fluid entering into the system. Earlier studies [94, 93, 92] have shown that the fluid modeling can be applied to model self-similar like traffic to a reasonably good extent. This study adopts single buffer stochastic fluid analysis proposed by D. Mitra [55]. A stochastic fluid system models the input as a continuous fluid, entering and leavening a storage device (called a buffer), according to randomly varying rate governed by a generating source. During the ON time, the source generates fluids at certain rate, \( \gamma \), into the buffer and otherwise. The fluid at the buffer is drained by a server at processing rate, \( f \); thereby the buffer can be empty when fluid generation-rate is low or zero. The generating source of fluids can be modeled as Continuous Time Markovian Chain (CTMC) and Discrete Time Markovian Chain (DTMC) (by uniformization of CTMC). Here, incoming fluid corresponds to the workloads sent to the system, fluid at the buffer corresponds to the remaining system workloads, and process rate of buffer is the operating frequency of server. Essentially, the stochastic fluid analysis can provide transition probability of the state of the system at each control point for building up
optimal frequency control rule by MDP.

Finite number of control points shown in equation 6.1 facilitates finite horizon MDP formulation to minimize energy consumption and maintain system performance. System performance metrics such as response time, can be presented in the closed form function of remaining system workloads. Therefore, this chapter directly uses the state of the system, $x$, as performance measures as shown below:

$$E(W_t, f) \leq \bar{W} \implies E(x_t, f) \leq \bar{X}$$

To establish the optimal structure of frequency control and ease of computation, $E(x_t, f) \leq \bar{X}$ is converted into the concept of feasible set of $A'$, defined as following:

$f \in A'$ such that $E(x_t, f) \leq \bar{X}$

The remaining system workload, $x$, is a decreasing function of frequency. More stringent the performance constraint is, higher the frequency should be allocated. The feasible action space, $A'$, decreases with the increase in remaining workloads, $x$. With such transformation, existing solution algorithms, such as policy iteration, can compute the optimal decision rules in a very efficient manner. In addition to that, this study establishes the optimal structure through such MDP formulation. Essentially, the optional policy is proved to be a threshold policy, in which frequency should increase with the state of the system (remaining workload in the system).

Two types of modeling, CTMC and DTMC, of stochastic fluids are introduced to derive the transition probability of MDP first in the subsection 6.2.1. Following that, rigorous MDP formulation is described in section 6.2.2 and the structure of optimal control frequency is studied in 6.2.3.
6.2.1 Fluid approximations for server workloads

Assuming web server traffic is governed by an ON-OFF source generating all the workloads into a single buffer and single server system as shown in figure 6.2. When the ON-OFF source at time \( t \) (denoted by \( z(t) \)), is at ON state \((z(t) = 1)\), it sends the traffic fluid into the buffer at rate \( \gamma(t) \) bytes per unit time, and does not send any traffic during the OFF state \((Z(t) = 0)\). The buffer here has infinite waiting capacity and a controllable server channel capacity, \((f_1, \ldots, f_n)\), to complete workloads in the buffer, denoted by \( x(t) \). When the snapshot is taken in the above figure 6.2, the channel capacity is serving a fixed amount \( f_i \) bytes per second to drain out the fluids.

Figure 6.3 shows how dynamics of fluid at the buffer is governed by an ON-OFF source in figure 6.2. In reality, only the fluid level (remaining workload) can be observed, but not the dynamics of ON-OFF source. Hence, the frequency control decision can be made following the fluid level (remaining workload) at decision epochs. For example, the threshold-based control can specify frequency, \( f_1, f_2, \ldots f_8 \), corresponding to fluid level of \( x_1, x_2, \ldots x_8 \). The ON-OFF source is essentially a modeling technique for capturing the dynamics of state of the system.

The transition between ON and OFF states of fluid generator can be modeled as CTMC by applying analysis from [55] and DTMC by uniformization of CTMC. Note that since stationary
traffic process is considered in this work, $\gamma(t)$ remains constant during every ON time.

6.2.1.1 CTMC

The generating source has ON time and OFF time following a renewal process and their durations are exponentially distributed with mean $\alpha^{-1}$ and $\beta^{-1}$ respectively. The process of generating source can be modeled as a CTMC with ON and OFF states.

The CTMC generator matrix, $M$, per unit time is as following

$$
M = \begin{pmatrix}
-\beta & \beta \\
\alpha & -\alpha
\end{pmatrix}
$$

Since the fluid is generated at rate $r$ during the ON time and 0 at OFF time, and in between it is drained out at rate $f$, the buffer is governed by the following draft matrix, $D$,

$$
D = R - fI = \begin{pmatrix}
0 & 0 \\
0 & \gamma
\end{pmatrix} - fI = \begin{pmatrix}
-f & 0 \\
0 & \gamma - f
\end{pmatrix}
$$
Essentially, the fluid level, $x$, changes according to the state of $z$, and choices of $f$.

$$\frac{dx(t)}{dt} = \begin{cases} 
-f & \text{if } z(t) = 1 \\
-f & \text{if } z(t) = 0, \text{ and } x(t) > 0 \\
0 & \text{if } z(t) = 0, \text{ and } x(t) = 0 
\end{cases}$$

Transition probabilities are therefore defined as following $H_{i,j}(t, x; x_0) = P\{x(t) \leq x, z(t) = j | x(0) = x_0, z(0) = i\}$, where $z = \{0, 1\}$.

$$H(t, x; x_0) = \begin{pmatrix} H_{0,0}(t, x; x_0) & H_{0,1}(t, x; x_0) \\
H_{1,0}(t, x; x_0) & H_{1,1}(t, x; x_0) \end{pmatrix}$$

The matrix $H(t, x; x_0)$ satisfies the following partial differential equation (PDE):

$$\frac{\partial H(t, x; x_0)}{\partial t} + D \frac{\partial H(t, x; x_0)}{\partial x} = MH(t, x; x_0), \quad (6.2)$$

with initial conditions

$$H(0, x, j; y, i) = \begin{cases} 
\begin{pmatrix} 1 & 0 \\
0 & 1 \end{pmatrix}, & \text{if } x_0 \leq x; \\
\begin{pmatrix} 0 & 0 \\
0 & 0 \end{pmatrix}, & \text{if } x_0 > x. 
\end{cases}$$

The boundary condition in equation 6.2 is saying at $t = 0$, $H_{i,j}(0, x; x_0) = P_{i,j}(x(0) \leq x)$ actually equals to $P_{i,j}(x_0 \leq x)$. The matrix of transition probability, $H$, at $t = 0$ only depends on the initial fluid value, $x_0$. It is not a trivial task to solve PDE as shown in 6.2. Ren and Kobayashi [71] used double Laplace transform method to obtain $H$, and reduced the PDE problem in 6.2 to the eigenvalue problem detailed in the following paragraph. They provided the closed form
solution, given in terms of explicitly identified eigenvalues and eigenvectors for infinite buffer case as following:

\[
H^*(s, x) = \begin{pmatrix}
H_{0,0}^*(s, x) & H_{0,1}^*(s, x) \\
H_{1,0}^*(s, x) & H_{1,1}^*(s, x)
\end{pmatrix} = L_x\{H(t, x)\} = \int_{0}^{-\infty} e^{-st}H(t, x)dt
\]

They expressed \(H^*(s, x)\) in the following equations, where (i) \(U(\cdot)\) denotes the unit step function, (ii) \(u(s), V(s)\) and \(U(s)\) are the eigenvalue, right and left eigenvector of \(D^{-1}[M' - sI]\), and (iii) \(e_{i_0}\) is a unit vector with its \((i_0 + 1)\) entry being 1 and all other entries being 1. Also, \(U_k(s)\) and \(V_k(s)\) satisfies the following normalization, \(U_k(s)DV_l(s) = \delta_{k,l}\), where \(\delta_{k,l}\) is Kronecker delta. \(^2\)

\[
H^*(s, x; x_0, z_0) = \{G_{N+1}(s) + \sum_{k=1}^{N-\lfloor c\rfloor} [a_k(s) + b_k(2)] \cdot V_k(s)e^{u_k(s)x} \}U(x - x_0)
+ \sum_{k=0}^{N} a_k(s)V_k(s)e^{u_k(s)x}[U(x) - U(x_0)]
\]

where

\[
G_{N+1}(s) = [g_{N+1,0}(s) \ldots g_{N+1,N}(s)] = -\sum_{k=0}^{N} \frac{V_k(s)U_k(s)}{u_k(2)} e_{i_0}
\]

\[
a_k = U_k(s)D'H^*(s, 0), 0 \leq k \leq N
\]

\[
b_k = \frac{U_k(s)d_{i_0}}{u_k(s)} e^{-u_k(s)x_0}, 0 \leq k
\]

Note that the subscript, \(k\), in above equations is index for eigenvalues and obeys the following conventions:

1. \(k \in \{1 \leq k \leq N - \lfloor c\rfloor\}\), corresponds to \(N - \lfloor f\rfloor\) negative eigenvalues, i.e., \(\text{Re}\{u_k(s) < 0\}\)

---

\(^1\text{Ut} - t_0 = 1, \text{if } t \geq t_0; 0, \text{otherwise}\)

\(^2\text{}\delta_{i,j} = 1, \text{if } i = j; 0, \text{otherwise}\)
for all \( \text{Re}\{s\} \geq 0 \)

2. \( k \in \{N - \lfloor f \rfloor \leq k \leq N\} \) corresponds to \( N - \lfloor f \rfloor \) negative eigenvalues, i.e., \( \text{Re}\{u_k(s) > 0\} \)
for all \( \text{Re}\{s\} \geq 0 \)

3. \( k = 0 \) corresponds to \( u(0, s) \), where \( \text{Re}\{u_0(s)\} \) for all \( \text{Re}\{s\} > 0 \), but \( u(0, 0) = 0 \).

Since \( H^*(s, 0) \) is not straightforward to obtain for \( k \in 1 \leq k \leq \lfloor f \rfloor \), Ren and Kobayshi [71] also provided extra derivation of obtaining close form presentation of \( a_k \) through \( b_k \) in [71]. The closed form of \( H^* \) eventually is very complicated and make analytically inverting \( H^* \) intractable. This study seeks for numerical inversion of \( H^* \) by existing fourier transform algorithms, detailed in the experiment section 6.3.

### 6.2.1.2 DTMC

The generating \( M \) matrix in CTMC modeling can be used to build up a DTMC model for the generating source by uniformitarian techniques. In order to build up DTMC model for generating source, the transition probability, \( Q \), of ON-OFF state during a unit time needs to be established. The goodness of transforming from \( M \) to \( Q \), depending on choices of parameter, \( c \), mapping continues time into discrete time point, as detailed in the following.

At time \( t \), the state of off-on source, \( z = \{0, 1\} \), is governed by the transition probability matrix, \( Q \),

\[
Q = \begin{pmatrix}
Q_{0,0} & Q_{0,1} \\
Q_{1,0} & Q_{1,1}
\end{pmatrix}
\]

The uniformization of the transition of generating source from CTMC can be viewed as an equivalent process, in which the system state is observed at random times which are exponentially distributed with parameter \( c \). The choices of \( c \) is significant in deriving transition probability based on CTMC and DTMC models. The details of uniformizing \( M \) matrix in CTMC into \( Q \)

\(^3\lfloor f \rfloor \) denotes the integer part of \( f \)
matrix is down below as shown [68]:

\[
\begin{pmatrix}
-\beta & \beta \\
\alpha & -\alpha
\end{pmatrix} \Rightarrow Q =
\begin{pmatrix}
1 - \frac{\beta}{c} & \frac{\beta}{c} \\
\frac{\alpha}{c} & 1 - \frac{\alpha}{c}
\end{pmatrix}
\] (6.7)

where, \( \sup_{s \in (\alpha, \beta)} s \leq c \leq \infty \).

Once the \( Q \) matrix of ON-OFF transition probability of unit time, the transition probability over \( \tau \) units of decision epoch can be derived by enumerating all the possible combination of \( \tau \) ON-OFF changes. The exact formula of transition probability over \( \tau \) is provided in MDP formulation at the next subsection.

### 6.2.2 MDP formulation

With underlying traffic modeled by ON-OFF fluids, we can formulate the original problem as a finite MDP problem. The definitions for state, action, transition probability, optimality equations and reward functions are given below:

- **State and state space**
  
  Let \( S_t = (x_t, z_t) \) be the state at time \( t \), where \( x_t \) is amount of fluid in the buffer at time \( t \), and \( z_t \) is the state of ON-OFF source at time \( t \).
  
  \( s_t \in \{0, 1, 2 \ldots \} \times \{0, 1\} \)

- **Action and action space**
  
  Let \( a_t \) be the action at time \( t \). \( a_t \in A = \{f_1, f_2, \ldots f_n\} \)

- **Cost function**
  
  \( r_{a \in A'}(s, a) = r_{a \in A'}((x, z), a) = c \cdot a^3 \), where \( c \) is the energy coefficient and \( A' \) is the feasible action set obeying performance constraint.

- **Decision epoch**
  
  \( t = \{1, 2, \ldots N\}, N \leq \infty \).
The decision epoch takes \( \tau \) units of time.

**Transition probability**

According to modelings of ON-OFF source, CTMC and DTMC, two sets of transition probabilities can be derived.

- **DTMC**
  
  Applying \( Q \) in equation 6.7 for transition of fluid ON-OFF state, this study approximates transition during \( \tau \) units of time by assuming \( \tau \) transitions of ON-OFF states. \( P_{i,j} \) can therefore by summation of probabilities of possible sequences, that has \( z(0) = z_i, z(\tau) = z_j \), and \( x_j = x_i + \sum_{k=1}^\tau z_i \cdot r \). Since \( z \in [0,1], \sum_{k=1}^\tau z_i \) can be redefined as \( l \) units of ON time taking places in \( \tau \) units of time. The exact formulation is listed as following.

  \[
P_{i,j} = \sum_{l=0}^{\left|x_i - x_j\right|} C^l(i,j)
  \]

  where

  \[
  C^l(i,j) = \sum_{v \in \delta} \left( \prod_{k=1}^\tau M_k^v \right)
  \]

  \( M_k \) is the transition probability of \( P(z(k)|z(k-1)) = Q(z(k)|z(k-1)) \), and \( \delta \) is the subset that \( z(0) = z_i, z(\tau) = z_j \), and there are exactly \( l \) units of ON time.

  where \( l \) denotes number of unit time the source is on amount \( \tau \) unit of time

- **CTMC**
  
  As shown in CTMC fluid modeling in section 6.2.1, we can obtain \( H\{\tau, x; x(0)\} \) by by numerically inverting \( H^* \). In order to \( P_{i,j}, x_i, z_i, x, \) and \( t \) are first substituted by \( x(0), z(0), x_j \) and \( \tau \) into \( H\{\tau, x; x(0)\} \)
\[ P\{ (x_j, z_j)|((x_i, z_i), a) \} \]
\[ = H\{ x(\tau) \leq x_j, z(\tau) = z_j| (x(0) = x_i, z(0) = z_i), a \} \]
\[ - H\{ x(\tau) \leq x_{j-1}, z(\tau) = Z_j| (x(0) = x_i, z(0) = z_i), a \} \]
\[ = H_{z_i, z_j}(\tau, x_j; x_0) - H_{z_i, x_j}(\tau, x_{j-1}; x_0) \]  

(6.8)

- **Performance Constraint**

\[ E(x) \leq \bar{X} \]

As mentioned earlier, in order to make MDP more trackable in building up structure for optimal policy and ease of computation, this performance constraint will be converted into feasible set of \( a, A' \). Action at every time point has the following equation \( a \in A' \) satisfies

\[ \sum_j P\{ (x_j, z_j)|((x_i, z_i), a) \} \cdot x_j \leq \bar{X} \forall i \]

- **Optimality Equations**

\[ V_n(x, z) = \min_{a \in A} \{ r((x, z), a) + \sum_{x'} \sum_{x' \in S} p((x', z')|(x, z), a) v_{n+1}(x', z') \} \]

The value function of \( t = n, V_n \), is the minimum of sum of current cost \( r((x, z), a) \) and expected future values \( (V_{n+1}) \) function. Decision rules for set of actions achieving the optimality equations can be claimed as optimal one.

Please note that this study eventually discretizes the fluid levels for the translatability of the models and solution methodology, even though the fluid is supposed to be continuous. The greater range of fluid levels are considered, the better are the proposed approximation, model, and solution methodologies.
6.2.3 Structure of optimal policy

It is intuitive that the frequency should increase with increasing system load in order to sustain performance. This study first shows that for the finite horizon DTMC modeling the optimal action function \( a(x, z) \) achieving the optimality equation is nondecreasing in \( x \), for given \( z \) (assuming \( z = 0 \) means fluid OFF and \( z = 1 \) means fluid ON). This optimality is derived through the submodular property of optimality equation, \( V_n \). For ease of understanding the following proof, state space of \( V_{n+1} \) is explained as follows. At time \( n = t \) control time point, the current state is \((x, z)\); control action, \( a \), is assumed to be taken, and there will be \( \tau \) units of time to \( n = t + 1 \) control time point. Essentially, the state of \( x \) at \( n + 1 \) depending on (i) the fluid is ON or OFF at \( t = n \), and (ii) how many ON times take place for the rest of \( (\tau - 1) \) units. Therefore, \( x(n + 1) = x_n + a + zr + lr \), where \( l \) is defined subsection 6.2.1 as the \( l \) units of ON for the rest of \( (\tau - 1) \) units.

Prior to deriving the proportions, three technical Lemmas are introduced below:

**Lemma 6.2.1.** (i) If \( f(x, y) \) is jointly convex in \((x, y)\), then \( \min_y f(x, y) \) convex in \( x \).

*Proof. (i)* Since \( f(x, y) \) jointly convex in \((x, y)\), \( RHS = f(x_3, y_3) \leq \lambda f(x_1, y_1) + (1 - \lambda)f(x_2, y_2), \forall y_1 and y_2 \)

Let \( g(x_3) = \inf_y f(x_3, y) \), then

\[
g(x_3) = \inf_y f(x_3, y) \leq RHS, \forall y_1 and y_2
\]

\[
g(x_3) \leq \lambda \min_y f(x_1, y_1) + (1 - \lambda) \min_y f(x_2, y_2) \forall y_1 and y_2
\]

\[
g(x_3) \leq \lambda g(x_1) + (1 - \lambda) g(x_2)
\]

Therefore, \( g(x) = \min_y f(x, y) \) is convex in \( x \). \( \square \)

**Lemma 6.2.2.** (ii) If \( f(x) \) is convex in \( x \), then \( g(x, y) = f(ax + by) \) is supermodular, for all \( a \geq 0, b \geq 0 \). If \( f(x) \) is concave in \( x \), then \( g(x, y) = f(ax + by) \) is suppermodular, for all \( a > 0, b < 0 \).
Proof. (ii) See Theorem 2.3.6 in [87]

Lemma 6.2.3. (iii) If $g$ is submodular function on $(x, y)$ and for each $x \in X, \min_{y \in Y} g(x, y)$ exists, then $f(x) = \min \{y \in \arg \min_{y' \in Y} g(x, y)\}$ is monotone nondecreasing in $x$.

Proof. (iii) See Lemma 4.7.1 in [68]

Proposition 6.2.4. Let $G_t((z, x), a) = c \cdot a^3 + E\{V_{t+1}(z', x - a + zr + lr)\}$. It satisfies:

(i) $G_t((z, x), a)$ is convex in $(x,a)$ and $V_t(z, x) = \min G_t((z, x), a)$ is convex in $x$.

(ii) $G_t((z, x), a)$ is submodular for any given $z$.

(iii) For any given $z$, $a(z, x)$ nondecreasing in $x$

Proof. (i) By backward induction. For $t = N$, $G_N((z, x), a) = c \cdot a^3 + V_N(z, x)$. Since $a \in A'$, a convex set of $(x,a)$, $c \cdot a^3$ is convex in $(x,a)$. $V_{N+1} = 0$ is convex in $(x,a)$. $G_N$ is a linear combination of convex function in $(x,a)$ and remain convex in $(x,a)$. Applying Lemma 6.2.3 (i), $V_N = \min G_N$ is convex in $(x,a)$.

Assume $G_n((z, x), a)$ is convex in $(x,a)$ and $V_n(z, x)$ is convex in $x$ for $n = t+1, \ldots, N$.

$G_t((z, x), a) = c \cdot a^3 + E\{V_{t+1}(z', x - a + zr + lr)\}$. Since $V_{t+1}(z, x)$ is convex in $x$ and $(x - a + zr + lr)$ is a linear combination of $(x,a)$ for any given $z, V_{t+1}(z', x - a + zr + lr)$ is convex on $(x,a)$, Expectation still preserves the convexity of $V_{t+1}(z', x - a + zr + lr)$ in $(x,a)$. Again, since $G_t((z, x), a)$ is a linear combination of functions convex on $(x,a)$, $G_t((z, x), a)$ is convex on $(x,a)$. Since $V_t(x, z) = \min_a G_t((z, x), a)$ and $G_t((z, x), a)$ is convex in $(x,a)$, $V_t(x, z)$ is convex in $x$ followed by Lemma 6.2.3 (i).

(ii) From (i), $V_{t+1}(z, x)$ is convex in $x$. Also, with Lemma 6.2.3 (ii), $V_{t+1}(z, x - a + zr + lr)$ is submodular, for any given $z$. $c \cdot a^3$ is also submodular. Since the transition probability for
given \((x, a)\) and \(z\) only depends on \(z'\),  
\[
G_t((z, x), a) = c \cdot a^3 + E\{V_{t+1}(z', x - a + zr + lr)\}
\]
is submodular.

(iii) It’s straightforward followed by (ii) and Lemma 6.2.3 (iii).

6.3 Experiments

6.3.1 Experimental set up

This section details about experimental setup and evaluation conducted to verify the design mentioned in last section. A simulator program, written in MATLAB, has been used for all the experiments. The workload used for this evaluation were of two types, namely, synthetic and real traces. Synthetic traces were used to compare CTMC and DTMC within the proposed model; whereas the real traces were used for extensive evaluation to test the effectiveness of the proposed methodology. The former was generated using exponential distribution with a workload-generator (written in MATLAB) using the two parameters - \(\alpha, \beta\) (the ON and OFF-time parameters, respectively) and \(r\) (fluid generating rate). Real traces were obtained from public domain in Internet, during the months of September and October 2004 [4]. Performance metrics considered for evaluation with synthetic traces was the amount of remaining workload (to be bounded within certain threshold). On the other hand, response time was the chosen metric for the real traces, for which the proposed methodology has to be built up by approximating the incoming workload as a continuous stream of fluid. The fluid stream has been discretized and normalized by forty (40) different levels as shown in table 6.3.2. Eight different levels of operating frequencies have been used to evaluate energy-performance control using two length of decision epochs (\(\tau = 5, \text{ and } 15\)) for proposed DVS schemes and existing algorithms.
### Table 6.2. Parameter and values for Synthetic Traces

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.352</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.65</td>
</tr>
<tr>
<td>$r$</td>
<td>0.8</td>
</tr>
<tr>
<td>$\tau$</td>
<td>5, 15</td>
</tr>
<tr>
<td>$\bar{X}$</td>
<td>18</td>
</tr>
<tr>
<td>frequency range</td>
<td>0.36, 0.41, …, 0.71</td>
</tr>
<tr>
<td>fluid range</td>
<td>40 levels from 1 to 40 with even and discrete increment</td>
</tr>
</tbody>
</table>

### 6.3.2 Compare with DTMC and CTMC

In order to evaluate the pros and cons of DTMC and CTMC, this study first shows how both the models approximate the transition of the system with different initial fluid values and time points. Since the difference of DTMC and CTMC lies on the modeling the transitions within a decision epoch, the length of the decision epoch can significantly impact the performance of both the models. Experiments with different values of $\tau$ (the epoch length) were conducted for identifying the merits of each model. Finally, workloads of different traffic intensities were generated to identify the threshold structures for optimal policies. The parameters used in the following subsections are listed in table 6.3.2.

#### 6.3.2.1 Approximation of system transition

Both of the fluid models, CTMC and DTMC, are meant to capture the transition of the system over time with the fluid generated using $\alpha$, $\beta$ (shown in table 6.3.2) and $\gamma$. Figure 6.4 (a) shows how CTMC models the transition of $P(x \leq 10, z = 0)$ over time when the beginning of the system is at “off” and $x_0 = 1, 3, 7, 10$. As for fixed interval, say $\tau = 5$, the DTMC demonstrates how the transition of $P(x(t) = j|x(0) = i)$ with different values of $i$ (1, 5, 10, 15, and 20) to all $j$, (1, 6, 11, 16, 21) as shown in figure 6.4(b).

From figure 6.4 (a), it can be seen that the system stabilizes significantly after $t = 20$, regardless of initial fluid values. It provides the insight on the length of control interval, pointing out if steady-state or transient analysis is more suitable. For this specific fluid parameters,
the transient analysis is more appropriate to serve as modeling ground for DVS control policy, considering control epoch less than 20 units. Moreover, smaller the difference between $x_0$ and $\bar{x}$ is, the faster the system converges to steady state. The other factors behind such scenarios might stem from the numerical transformation of Laplace function. Since the $P(x < \bar{X})$ is obtained from the numerical Laplace inversion (known to be very sensitive to the initial periods and final
values of $x$, such as small $t$), the cumulative probability from CTMC might not be that stable under such cases.

As far as $P(x(t = 5) = j|x(0) = i)$ is concerned, it shows similar patterns for different $i$’s to all $j$’s. This fact shows that the transitions within small control intervals is very regular, because of the discretization of fluid levels. Also, the stability condition of the system ($\tau < 1$) can limit the changes in the state of system within a short time span. Actually, the computation of system transient behavior by DTMC increases with increase of value for $\tau$. B. Serocola [81] actually drives a computation algorithm to do the fast computation based on the concept of DTMC to make transient analysis tractable for large $t$.

### 6.3.2.2 Varying the parameters

Prior to showing the effectiveness of proposed methodologies, system with fixed operating frequency was simulated as comparison basis, shown in figure 6.5(a) and (b) with different fluid generation rate, $r = 1, 1' = 1.5$. If the workload threshold($\overline{X}$) is set to 18, running at maximum frequency is the only fixed frequency policy which can achieve that. On the other hand, implementations for both DTMC and CTMC frequency control can achieve the performance constraint ($X \leq 18$) but operates at lower frequency during the lower loads to save energy as shown in the figure 6.6, 6.7, 6.8 and 6.9. From those figures it can be seen that optimal policies derived form CTMC are slightly more conservative than DTMC for all the cases, and therefore results in consuming lower energy. The performance of both the models also vary with different values of $\tau$ and $r$. Note that as mentioned in chapter 5, the utility of DVS should be more emphasized with respect to performance sustainance because its energy saving was limited by the server allocation, and requests scheduling. Hence, both of these DVS schemes can control fluid traffic very well as evident from figures (6.6, 6.7, 6.8 and 6.9).
Figure 6.5. System Performance with Fixed Frequency Control

(a) \( r=1 \)  
(b) \( r=0.8 \)

Figure 6.6. DTMC v.s. CTMC at \( \tau = 5, r = 1 \)
6.3.2.2.1 Effect of window size  For both of the models under fixed \( r \), more than two frequency levels have been considered in optimal policies. At the case of \( \tau = 5 \), only frequency \( f_1 \) and frequency \( f_8 \) are adopted and separated by certain threshold values of the state of fluid. It is so called “bang-bang” policy, which either operates either at the minimum frequency or at the maximum frequency. On the other hand, during \( \tau = 15 \), there are more choices of frequencies involved for optimal policy, because transition probabilities between states are more spread out, compared to those for \( \tau = 5 \) (shown in figure 6.4). Thereby, the energy savings is better explored when considered at higher \( \tau \) values for both of the models. When value of \( \tau \) is small, thresholds for optimal policies are computed based on CTMC, and DTMC models. This fact favors usage of DTMC modeling, which demands much less computation than CTMC in small decision epoch. And, CTMC model is suggested to be used with higher values of \( \tau \) to avoid high volume of
6.3.2.2.2 Traffic intensity This study considers two different fluid generation rates, \( r = 0.8 \) and \( r = 1 \), to investigate the possible effects of traffic intensity. Value of \( r = 1 \) can generate medium load into the system and \( r = 0.8 \) creates low traffic intensity, whose optimal policy involves fewer choices for frequencies. This matches with the discussion in chapter 4 and 5 that the benefits of DVS on energy savings can be explored in a better manner with medium load, than that with high or low load. Also, for any given fluid model, higher value of \( r \) results into higher optimal frequency selection, for any given fluid state. Thus, differences in threshold values for optimal frequencies (when \( r \) is varied) are not much varying. On the contrary, when \( \tau \) is varied, the differences in DTMC and CTMC show higher variance. Actually, when decision epoch is
small ($\tau = 5$) and traffic intensity is low ($r = 0.8$), both of CTMC and DTMC suggest the same decision rules for frequency control, which is again bang-bang policy. Observed results suggests that bang-bang policy is suitable for fine-grained frequency control during the low load.

### 6.3.3 Compare with existing DVS heuristics

This study demonstrates the relative performance between proposed (proactive) DTMC and CTMC DVS algorithms with two existing frequency control methods - IBM-ARL, and On-Demand. The IBM-ARL mechanism works in reactive mode and was proposed in [33]. The OnDemand algorithm is also a reactive solution used in Linux OS for frequency setting [61]. Ondemand method is traditionally more conservative towards maintaining performance. The
comparison metrics are response time, and energy consumption, which are shown in 6.3.3 and 6.3.3 with \( \tau = 4 \) and \( \tau = 16 \) time units. Here one time unit equals fifteen (15) seconds. Time granularity, investigated for frequency control are in units of 60 seconds and seconds. These are more refined than control granulations considered in chapter 4 and 5, because of low on-line computational overhead. For proposed DTMC and CTMC algorithms to work, the history data needs to be fit into the fluid model to obtain parameters of \( \alpha \), \( \beta \), and \( \gamma \). Accordingly, with suitable level of discrete levels of fluid, the optimal policies will be computed at the beginning of interval for server allocation and server-application mapping.

From table 6.3.3 and 6.3.3, following observations.

- Generally, as observed in earlier chapter, IBM-ARL puts more preference to save energy while Linux-Ondemand algorithm is more conservative towards performance. Methodologies proposed in this chapter adopts a middle path between these two to obtain a happy balance between performance benefits and energy-savings.

- Performance-wise, IBM-ARL and Linux-Ondemand methods are very sensitive to the control epoch length. When \( \tau \) is low, the average response time is lower because of more information of the state of the system. But it’s too sensitive, so that the system become very jittering. On the other hand, the proposed DVS policies can be interchangeably used for different values of \( \tau \) to accommodate changes.

- Overall, DVS is very critical to ensure performance within certain bounds. However, often
the savings on energy is not significant enough. Hence, DVS methods can be looked on as a good performance-sustaining methodology, with marginal improvements in energy-savings. However, since performance is sustained with along-with savings of energy, DVS should be preferred over having no frequency-control policy at all. Nevertheless, it often provides only marginal benefits, with given set of resource allocation, when traded off with energy-savings.

6.4 Conclusion Remark

This chapter proposed and evaluated two frequency-control/DVS approaches, which are easily implementable in an online fashion. Both of them utilize the information of remaining workloads and execute frequency control, following simple threshold rules. They not only the ensure the scalability but also responsiveness of the system by considering the feedback from the system. The savings on energy and benefits of performance are factors to be traded off in the design of proposed DVS policy. From the viewpoint of performance, the proposed schemes are more performance-friendly than saving energy. Comparison with other DVS schemes reveals that the amount of energy saved is not much higher, either.

In addition to merits in implementation, this study also provides two-fold insights for frequency control over web servers: (1) approximation of fluids can provide a sound solution for transient analysis of self-similar traffic. Since the transient analysis models the length of the decision epoch, the proposed DVS schemes can benefit significantly both on energy-savings and achieved performance in any defined decision epoch. Among the proposed fluid mode schemes, CTMC is suggested for longer control epoch whereas DTMC is better for finer granularity because of lesser computation involved. However, both of the strategies can be used interchangeably depending on the length of decision epoch. (ii) Combined with the fluid modeling, this study establishes the structure of the optimal policy as threshold type, which increases with the re-
maining workloads in the system, based on MDP formulation. The optimal policies obtained in evaluation experiments also supports the argument.
Power-performance management has been an ever important issue for the web hosting center (a multi-server queueing system, serving multiple application of a disparate nature). The high energy consumption of such a system is attributed to the conservative approach of resource-provisioning, which is configured to handle the worst-case behavior. Often the resources are provisioned to sustain the performance guarantee agreed upon in the SLAs, for applications under time-varying self-similar like traffic, whose performance measures are much more complicated to obtain than the standard queueing traffic. Prior studies have either investigated one of these aspects at a time or have considered a small subset of resources (with simplifying assumptions) that go into the control design. The goal of this study is to design an energy-aware management mechanism with multiple resources that minimizes the energy consumption as much as possible on one hand, while adhering to the performance-guarantees specified in SLAs on the other hand. To that end, this study proposes a multi-tiered solution methodology, in which three knobs of resources are considered. These knobs are number of servers to be used, their operating frequencies, and request-to-server mapping. According to the availability and suitabil-
ity of control resources, three tiers have been designated to work at system, application, and server level and they have been referred to accordingly. At each of the tier, this study performs tasks of traffic prediction, performance measures, and control optimization design by operations research techniques. The proposed methodology has shown significant improvements in energy saving and sustainance of performance guarantees at each level, as opposed to the comparable power-performance management schemes proposed earlier.

At the system level, the focus of the study is on provisioning the resources across applications with time-varying and self-similar like workloads at various control granularities. The system is modeled as a multi-server queueing system with a consideration of the wear-and-tear costs for servers, which was lacking in prior studies. A mixed nonlinear integer optimization is framed with respect to the two coupled control variables, the servers provision and the centralized server frequency control (among the applications with respect to control time points). This study proposes two solution methodologies, the SQNO and the Hybrid, which are suitable for different control granularities for the two resources. SQNO is a proactive optimization algorithm, which is based on $G/G/m$ steady-state performance modeling with predicted workloads as input. Compared to the feedback control approach [27], SQNO can be a better alternative for course-grained server allocations, but less responsive for fine-grained frequency control. The Hybrid strategy takes the best of both worlds, utilizing the predictive information of the SQNO at coarse time granularities for server provisioning, and applies the feedback control mechanism at short granularities for DVS. Moreover, the Hybrid scheme is also well suited for a more practical setting where the server provisions are performed less frequently due to the high time overhead for switching on servers or migrating applications and the DVS control can be applied more frequently. This leads to the development of a feedback-based DVS scheme as the third tier of the solution that models the feedback of the transient state of the system.

The proposed second tier of the solution consists of a methodology that packs/schedules the various types of services into a given set of servers to maintain the balance of energy and
performance for an individual application. It is presented as a constrained optimization problem, which is the first formulation of its kind (with that specification and goal). A two step solution is proposed to co-locate services sessions on these servers as a first step, and then conducting decentralized DVS at the local servers as the second step. The packing of services to the servers is posed as a bin-packing problem with the constraint of balancing the load distribution among servers (the mean and standard deviation specifically), which transforms the original stochastic problem into a robust optimization problem by the capacity assurance parameter. Moreover, this parameter can be used to trade-off the small loss in performance in lieu of significant energy reduction, especially for the case of heavy-tailed distributed workloads. The proposed solution scheme for the first step allows a tunable set of knobs - choice of server selection and constraint checking for bin-packing - which are able to achieve more stringent performance requirements with a lower number of servers than a naive packing strategy. The second step performs decentralized prediction-based pro-active DVS control to adjust the frequency/voltage of execution for power savings. It is shown to be a happy balance between energy saving and performance sustainance, compared to with other existing DVS algorithms.

Due to the effectiveness of feedback from the system state and complication associated in predicting the workload for DVS control, this study proposes and evaluates two frequency control/DVS approaches as the third tier solution. It utilizes and models the feedback about the state of the system as control inputs and therefore can be implemented in an online fashion. A Markov Decision Process (MDP) is formulated to achieve the minimization of energy and SLA requirement at each server. The underlying transition of workloads in the MDP are modeled by two approaches (CTMC and DTMC) based on on-off fluid modeling which is a reasonable approximation for self-similar like traffic. This study also proves that the optimal frequency is an increasing function of the state of the system (remaining system workloads) for a given initial fluid state. The threshold policy of frequency control based on the remaining workloads is claimed to be an optimal one by the formulation here. This result is also validated on synthetic traces by
DTMC and CTMC modeling. Moreover, CMTC modeling is shown to be the better alternative for frequency control in a longer decision epoch. On the other hand, the DTMC modeling works well for a very refined frequency control. In principle, both the approaches can maintain system performance well at the marginal increase on energy cost, compared to the existing DVS schemes.

The proposed three-tiered methodology at each level can be applied in a coupled or decoupled manner, depending on the specification of the applied systems. Even though this study was originally motivated by the power-resource management issues at web hosting centers, the proposed methodologies and analysis can be applied to other large scale multi-server queueing systems with multiple classes of customers. The energy function used in the objective can be generalized as a cost function; and performance constraints can be easily substituted according to the system specification. Thereby, this study not only has made contributions in the field of computer science by providing a three-tiered performance-power management for web hosting centers, but also in operations research by combining various techniques such as stochastic processes, non-linear integer programming and statistical prediction, to model self-similar like traffic. The inter-disciplinary nature of the approach makes this study significant in its scope for the applicability, coverage of the relevant issues, and completeness in offering the solutions.
Pageview services offered by current web-servers to multiple classes of clients aim to minimize client perceived response time and failure rate of connection establishment etc. following different Service Level Agreements (SLAs). Accepting all incoming requests often run into the risk of exhausting available resources, resulting in high server-end response time. On the contrary, unsophisticated admission control policy might incur high connection time or even, may increase failure to view a page. Many admission control mechanism [32, 16] have been developed to reduce the server-end response time. However, often these schemes have neglected the impact of rejection on connection time happening because of repeated SYN retransmission. The motivation of this work stems from the lack of a suitable analytical model specifying a robust and agile control mechanism which would be capable of drawing a balance between admission control and loss in revenue (because of admission control). This appendix presents a two-fold non-intrusive admission control strategy to optimize client-perceived response time across various classes of clients. Control policies based on steady-state analysis and transient state management explore the trade-offs required to achieve performance goals and enables accurate tuning of appropriate
control knob(s).

Requests from clients may be served by establishing connections with the server or may get rejected by the server due to limited server capacity or admission control policies. A connection establishment happens through a three-way handshake policy with TCP. From client side, a retransmission of SYN packets take place at the regular intervals of 3, 6 and 12 seconds respectively until acceptance of the connection happens at the server. Consequently, client perceived response time include time taken to successfully establish a connection (before entering the server system) apart from actual response time at server end. Because a failure in connection establishment is associated with loss in revenue, it is extremely crucial to leverage resources on server side to ensure minimization of both acceptance time and response time for accepted requests and overall drop-rate of incoming requests.

Existing pageview response time management can be decomposed into three components; namely, connection time, server time and transfer time. Each of these can be separately controlled by different mechanisms. Studies have shown that effective modification on the top current TCP SYN transmission mechanism can reduce connection time by a significant extent. There are two possible approaches to reduce server-end response time. One is to modify existing three-tier web servers architecture and tune internal system parameters, such as scheduling policy or service rate of each tier. Not only this approach entails huge amount of effort, but also it fails to achieve the desirable in a transparent manner. Modification of application may expose the application to be exploited in various malicious ways, too and hence, may not provide a much-appreciated solution. The other approach is intrusive in the sense that it changes the load, injected into the system, by management of SYN packets, which is crucial to manage connection time as well, and object size management.

With these motivations, this appendix proposes a two fold non-intrusive admission control (NIAC) strategy to optimize multi-classes clients’ perceived response time. NIAC first tracks pageview statistics and build up optimization for multi-class pageview perceived response time.
based on a decomposed pageview models, which includes a serve-end queueing network model calibrated with only observed response time as input. NIAC is built on ksnifer [59], a kernel-based traffic monitor that accurately estimates pageview response times as perceived by remote clients. Secondly, NIAC employees two types of control mechanisms, (i) fast SYN and SYN/ACK retransmission, and (ii) embedded object removal and rewrite to achieve optimization results obtained from the first phase by observed arrival rate.

Analytical results from NIAC shows that optimal SYN management over pageview connection should (i) drop the second and the third SYN of the first connection if the first SYN of the first connection is dropped, (ii) accepted the first SYN request of consequent connections for download embedded objects. Essentially, optimal non intrusive admission control lies on the first SYN of the first connection, whereas the consequent connections from the same page will be served. Transient control mechanisms from NIAC maps the optimal steady state arrival rate of first connection to collectable system statistics as an indicator to trigger NIAC control options. The experiments results are in general effective and show different pros and cons in the proposed transient control mechanisms.

The rest of this appendix is organized as follows. The next section reviews related work. The proposed decomposed model for perceived pageview is presented in sections A.2. Analytical results to optimize multi-classes client perceived response time and transient control design is describe in section A.3 respectively. Section A.4 contains the experimental results and section A.5 concludes with the contributions.

### A.1 Background and System Overview

A pageview service in this appendix is referred to the rendering of the webpage in its entirety at the client browser. Container page and multiple embedded objects under the same page are retrieved sequentially at the client end through multiples connections requested to the server. For
example, Figure A.1 shows an example of how client requests service of pageview with index.html and 8 embedded objectives. Once the container page is fetched on the first connection, clients’ browser downloads 4 embedded objects along that connection and meanwhile requests the second connection to download remaining 3 embedded objects (namely, obj3.gif, obj6.gif and obj8.gif). A perceived clients’ preview response time is therefore initiated at $t_0$, when the first SYN of the first connection was sent, and ends at the $t_e$, when the last embedded object is retrieved.

At the server end, TCP ensures flow control to prevent overload by dropping SYN through well-known TCP exponential backoff mechanism. The process has been depicted in figure A.2, where there is the case of 2 SYN drop. The server drops the initial connection request originated from client at time $t_0$. TCP at the client side retries after 3 seconds with a second SYN packet, which may be accepted or dropped. If that is dropped too, then the next SYN retransmission occurs at time $t_0 + 9s$. The timeout period doubles (from 3s, 6s, 12s, etc.) until either the
connection is established or the maximum number of SYN retries is reached. Because of this fact, server SYN drops for even a single connection may delay the entire process and increase the client-perceived response time.

Further, if the first connection of a pageview is still not established after TCP maximum timeout time, a client browser considers it's a pageview failure associating with physical 21s waiting. Depending on classes' service agreement, clients may perceive different values for failure pageview response time. Pageview failure can also happen due to connection failure on the second and the third connection, after the first connection get established. In such a case, pageview failure will be reported by TCP after 21 seconds and the browser indicates the end of the page download at $t_z + 21$, although all objects successfully obtained from the server are obtained over the first connection during the time interval $t_0$ through $t_x$.

Our system attempts to alleviate problems associated with a pageview service as noted above. To that end, it employs a non-intrusive online model, which is a stand-alone appliance, located at the server-end, between network and server. It operates as a server-side mechanism with a low-delay control path to the web server that is unaffected by outside network conditions. Its
non-intrusive nature is because of the fact that it does not require any modifications either to the application hosted at the server-end, browser at the client-end or server complex. In fact, it is designed as a set of dynamically loadable kernel modules which operates above the network device independent layer inside the OS.

As a middle layer between server and client, NIAC provides the following two functionalities, (i) passively capture the network packet to measure the client perceived response time based on the early work [59] and (ii) actively manipulate the packet stream between client and server. The admission optimization module residing in NIAC uses clients’ real-time information, such as perceived response time, concurrent active connection, arrival rate and etc, as feedback to guide the management of packet streams. Following given policy from optimization module, the NIAC is able to throttle SYN onto NICA buffer and also perform a SYN retransmission technique, which retransmits the SYN from the stack, on behalf of the remote client. Figure A.3 depicts the behavior of the NIAC SYN throttling and retransmission. Consequently, Connection time and server response time of pageview are regulated and driven by model. Such a modification of SYN packets won’t be considered as violation of TCP protocol, because NIAC is built in within the same complex as server and not transmit the SYNs over the network.
A.2 Client Perceived Pageview Model

We propose a decomposed pageview modeling approach based on the following assumptions about the system and notation conventions. The model can be adequately validated, as discussed in subsection A.1. The exact values needed for configuration are system dependent, such as object-downloading policy of Internet Explorer and Firefox.

- Assumption 1: Each page may lead to generation of multiple types of requests, such as requests for index.html/container object, embedded objects etc. Each type of requests can be distinctly different in size and amount of computational resources needed than that for others. For example, embedded objects have wider range of size distribution than index.html and because of that reason, we consider one type of html requests and multiple types requests for embedded objects (which may require similar amount of computational resources). Type for request for an object is refereed by subscript $k$; $Type_k$ can be $\{html, object_1 \ldots object_K\}$.

- Assumption 2: Each page requests the first connection to download the index.html page and then demands up to $I$ connections to download $F$ embedded objects either through the existing connection or establishment of new connections. Therefore, required response time of all $I$ connection is dependent on (i) processing time of index.html on the first connection and (ii) processing time of multiple types of embedded objects sent through that connection. Order of a connection from a pageview is denoted by $i$, ranging between $(1, \ldots I)$. When $i = 1$, it corresponds to the first connection to download the container page.

- Assumption 3: Establishment of a new connection has maximum of $J$ SYN transmissions, whose order is referred to by $j$ subscript, $j \in 1, 2 \ldots J$. Most of dominant Internet browsers has $J$ set equal to 3.
Table A.1. Notation and Explanations

With aforementioned assumptions and notations (and system specification in section A.4), we first show how to get the decomposed algebraic expression for individual pageview response time and then derive the average pageview response time.

Individual pageview response time

Client perceived per admitted page response time (RT) is a max function of required response time of all its \( i \)th connections, because a pageview is considered to be finished only with the completion of the last of all connections. Required response time of \( i \)th connection, \( RT_i \), can be decomposed into two parts, (i) html processing time, which is the summation of html page connection time (\( T_{\text{con}}^{\text{html}} \)), server time (\( T_{\text{ser}}^{\text{html}} \)), and transfer time (\( T_{\text{tran}}^{\text{html}} \)) and (ii) multiple embedded objects processing time on \( i \)th connection, which is again the summation of all objects’ connection time (\( \sum_k T_{\text{con}}^{\text{obj}} \)), server time (\( \sum_k T_{\text{ser}}^{\text{obj}} \)), and transfer time (\( \sum_k T_{\text{tran}}^{\text{obj}} \)). Mathematically, individual pageview response time (as perceived by client) can be expressed as the following equations.

\[
RT = \max_i \{ RT_i \}, \text{ where}
\]

\[
RT_i = T_{\text{con}}^{\text{html}} + T_{\text{ser}}^{\text{html}} + T_{\text{tran}}^{\text{html}} + T_{\text{con}}^{\text{obj}} + T_{\text{ser}}^{\text{obj}} + T_{\text{tran}}^{\text{obj}} + RTT
\]

\[
T_{\text{con}}^{\text{obj}} = 0, \text{ if } i = 1; \text{ otherwise}
\]

Note that connection time is only required at establishment of connection via three-way handshaking mechanism. When particular embedded objects are downloaded through the same
An example of a single pageview downloaded through 2 different connections can be graphically illustrated as figure A.4. It uses the first connection to process request for index.html and then open second connections to download 8 objects simultaneously along 5 objects being downloaded on the first connection. The required response time on connection 2 ($RT_2$) depends on the completion of request for index.html on the first connection; and therefore $RT_2$ needs to add the connection, server, and transfer time of index.html on the first connection. Since $RT$ is the max function of $RT_1$ and $RT_2$, inappropriate response time management of either connection deteriorate the individual pageview response time and therefore average pageview response time $RT$.

Such an approach for decomposed analysis of individual pageview response time is based on sequencing resources requirements for all types of objects. It readily paves the road for modeling...
and managing average response time \( E(RT^c) \) and failure rate \( \wp^c \) of class \( c \) in the context of multiple classes. Connection time is linked to server connection capacity, arriving SYN packets and their time-out values; whereas transfer time is dependent on size of objects being downloaded, RTT and network-loss in that particular connection. For server-end response time, we especially build up a three-tier model with respect to different types of objects, by using pageview SYN arrival rate as input to relate to connection time. We derive the algebraic expressions in the following two subsections.

### A.2.1 Average Pageview Response Time

Average pageview response time for admitted pages can be decomposed into average values of every component as shown in equation A.2 for a single pageview.

\[
E(RT^c) = \max\{RT_i^c\} \quad \text{where}
\]

\[
RT_i^c = T_{conhtml}^c + T_{serhtml}^c + T_{tranhtml}^c + T_{conobj}^c + T_{serobj}^c + T_{transobj}^c + RTT
\]

Average connection time can be modeled through initial SYN packets, SYN retransmission and associated time-out values at the connection level. Average transfer time can be determined by the sizes of objects downloaded through that connection, RTT and network, whereas the latter two are not under server-end control. Lastly, average server-end response time is the function of available service rate and admitted rate over multiple classes and multiple types. Due to the complexity of server end, we explain the multi-tier analysis for average server response time in subsection A.2.2 and connection time and transfer time in the following subsection.

1. **Connection Time** \( (T_{conhtml}^c, T_{conobj}^c) \)

Each page request will try to either reuse the existing connection or establish the first connection with maximum \( I \) retries. As for establishment of a new connection for downloading
index.html and objects, it’s executed through SYN transmission mechanism as explained in section A.1. We assume SYN of $i^{th}$ connection of $j^{th}$ retry from class $c$ first arrives at service states ($S_{html+object}$ with rate $\lambda_{i,j+1}$ from transmission states ($S_{i,j}^{c}$). A transition diagram of single class $C = 1$ and maximum 2 SYN retries ($J=2$) is shown in figure A.5. When $j = 0$, it denotes the initial SYN transmission and $\lambda_{i,0}$ present the initial external SYN arrival. The failure of the second SYN retry is indicated by $j = 4$. If the connection is a re-used existing connection ($j = 5$), there is no connection time added and it can directly go to $S_{html+object}$ without transiting through $S_{i,j}$. SYN packet of $i^{th}$ connection and $j^{th}$ retry stays at transmission state $S_{i,j}^{c}$ for $TO_j$. Current TCP exponential backoff mechanism takes $TO_j = \{0, 3, 6, 9\}$. Once $TO_j$ time is reached, SYN packet transits to either $S_{i,j+1}^{c}$ or $S_{html+object}$ with dropping probability $P_{i,j}$ and $(1 - P_{i,j})$, respectively. Therefore, $\lambda_{i,j}$ can be recursively expressed by $P_{i,j}$ and $\lambda_{i,0}$

$$\lambda_{i,j} = (1 - P_{i,j}) \prod_{k \leq j-1} (1 - P_{i,k}) \lambda_{i,0}, \forall j \leq m$$

$$\lambda_{i,0} = \sum \lambda_{i,j} \forall j \leq m$$

Connection time of admitted requests (finally entering to $S_{html+object}$) can therefore be a summation of weighted average of time spent in $S_{i,j}^{c}$.

$$T_{con}^{html} = \sum_{j=1}^{3} \frac{\lambda_{1,j}}{\lambda_{1,0} + \lambda_{1,5}} TO_j$$  \hspace{1cm} (A.3)

$$T_{con}^{obj_i} = \sum_{j=1}^{3} \frac{\lambda_{i,j}}{\lambda_{i,0} + \lambda_{i,5}} TO_j, \forall i > 1$$  \hspace{1cm} (A.4)

Note that $T_{con}^{obj_i}$ only occurs when embedded objects are downloaded through the second or higher order connections, which are established after the first connection ($i = 1$) for
index.html is established. Once SYN is transited into $S_{html+object}$, both container and embedded objects of a pageview is first processed with certain amount of sever time respectively and transfer time respectively. Therefore, when deriving $T_{ser}^c$ and $T_{tran}^c$, it needs to be associated with acceptance rate of connections, $P_{c}^i$, $1 - \frac{\lambda_c^i}{\lambda_c + \lambda_s}$.

2. Transfer Time $(T_{tran}^{html} + T_{tran}^{obj})$

Transfer time depends on RTT, network loss, and size of downloaded objects, which can vary across classes. When the network connection between the client and the server is the bottleneck, it can become a dominant factor over a pageview response time. Several analytic models of have been developed [60, 20, 86].

$$T^{tran} = f(size, RTT, loss)$$

Cardwell’s model predicts that under higher loss rates and longer RTT, reducing object
size can reduce can improve transfer time more efficiently. RTT and loss rate for each class are functions of the end-to-end path from client to server through the Internet and therefore, are uncontrollable. Hence, transfer time over connection $i$ can be controlled by average total size of all types objects, $\sum_{k=2}^{K} \theta_{k,i}^c S(c,k)$, where $\theta_{k,i}^c$ is average number of $k$ type objects downloaded on $i$ connection and $S(c,k)$ is the average size of type $k$ object from class $c$. Since there is only one container object to be downloaded on the first connection, $\theta_{1,1}^c$ is assumed to be 1. Values of $\theta_{k,i}^c, \forall k > 2$ and $S(c,k)$ are client specific parameters and can be obtained by statistical profiling of clients workload.

$$T_{tran}^{html} = P_1^c f(S(c,1), RTT^c, loss^c) = P_1^c \Gamma_1^c$$ (A.5)

$$T_{tran}^{obj} = \sum_k P_i^c f(\theta_{(k,i)}^c s(c,k), RTT^c, loss^c) = P_i^c \sum_k \Gamma_k^c$$ (A.6)

Note that under realistic Internet conditions (an RTT of 80ms and loss rate of 2% [102]), $T_{tran}^{html}$ can be well estimated by a constant value due to the similar size of index.html. Contrarily, since embedded objects have large size variation, $T_{tran}^{obj}$ is also varying with total object size ($\sum_k \theta_{(k,i)}^c S(c,k)$) downloaded on that connection. Moreover, $T_{tran}^{html}$ is smaller than $T_{tran}^{obj}$ in order because of the size difference. Good transfer latency management lies on the size management of embedded objects.

### A.2.2 Server-end Response Time

With our prior knowledge, a model of multi-tier web server architecture system is based on $t = 1, \ldots, T$ server tier to serve multiple classes of their multiple types of requests. Here, we specifically consider the case where $T = 3$, which consist of “Apache” as the first tier, “tomcat” as the second tier and “DB” server as the third tier, shown in figure A.6. Index.html of each pageview is associated with type $k = 1$, and it requires the computational resources from all
three tiers. Embedded objects are associated with $k = 2 \ldots K$, where $K$ is decided by the size distribution of embedded objects. Note there is only one class of embedded objects shown in the figure A.6 for the graphical explanations and it usually requires multiple types to match actual size distribution. All types of embedded objects require different amount of computational resources from Apache server.

We assume that effective arrivals of type $k$ and class $c$ are Poisson with rate $\Lambda_{c,k,t}$, entering the station $t$ of such three-tier system. Effective arrival rate is the function of total admitted SYN arrival rate ($\sum_i \sum_j \lambda_{i,j}^c$) and average number downloaded $k$ objects of class $c \sum_i \theta_{k,i}^c$. Service rate of station to process type $k$ objects from class $c$ follows the distribution with mean $\frac{1}{\mu_{c,k,t}}$. At station $t$, effective arrival rate, $\Lambda^t$, equals to the summation of arrival rates of all entering types to that tier ($\Lambda^t = \sum_c \sum_k \Lambda_{c,k,t}$). For example, $\Lambda^{Apache} = \sum_c \Lambda_{c,html} + \Lambda_{c,obj}$, where $\Lambda_{c,obj} = \sum_{k=2}^K \Lambda_{c,k}$. Each tier of servers are assumed to adopt process sharing scheduling policy. Since incoming requests are assumed to be executed in process sharing discipline, all queues are symmetric thus quasi-reversible, and have the product form. Hence, expectation of stationary response time of type $k$ from class $c$ at station $t$, $R(c,k,t)$, can be obtained as following: $E[R(c,k,t)] = \frac{\mu_{c,k,t}}{\mu_{c,k,t}(\mu^t - \Lambda^t)}$, and mean server response time of type $k$ and class cjob is $E[R(c,k)] = \sum_t E[R(c,k,t)] = \sum_c \sum_k \frac{\mu_t^c}{\mu_{c,k,t}(\mu^t - \Lambda^t)}$, where $\mu^t = \sum_c \sum_k \mu_{c,k,t}$. Therefore, $T_{html}^{ser}$ be expressed as following equations.
For an individual embedded object of type $k$ ($k \geq 2$) from class $c$, its average response time can be expressed as \( \frac{\mu_1}{\mu_1 - \Lambda_1} \). Total object server time will be the summation of individual $k$ type object server time over average number of $k$ object, \( \sum_i \theta_{k,i}^c \). Average server time of embedded objects on $I^{th}$ connection can be expressed as follows.

\[
T_{ser}^{obj} = P_c \{ \sum_k \theta_{k,i}^c \frac{\mu_1}{\mu_1 - \Lambda_1} \} \tag{A.8}
\]

Overall, we can write a general form of

\[
T_j = P_c \{ \sum_t \sum_k \theta_{k,t}^c \frac{\mu_t}{\mu_t - \Lambda_t} \}
\]

Combining equation A.3-A.8, we can express admitted $RT_c$ as following.

\[
E(RT^c) = \max \{ P_c \{ \sum_t \sum_k \theta_{k,t}^c \frac{\mu_t}{\mu_t - \Lambda_t} \} + \sum_j \Gamma_j \}
\]

\[
= \max \left\{ \frac{X}{\lambda_i,0 + \lambda_i,5} \right\}, \tag{A.9}
\]

### A.2.3 Failure Rate

If a SYN of any connection from a single page fails after the second retry, we considers it’s a page-view failure. Average failure rate of class $c$ at connection $i$ is \( \frac{X}{\lambda_i,0 + \lambda_i,5} \). Average failure rate
of class $c$, $\phi^c$, is $\max_i \frac{\lambda^c_{t,i}}{\lambda^c_{o} + \lambda^c_{o,t}}$.

### A.3 Control Mechanism

We consider a system with multiple classes, each demanding different levels of quality of service, defined in terms of metrics such as average response time and failure rate, with associated costs. The cost structure of each class has been modeled as: (i) $W^c$ is the dollar cost of response time per unit time for class $c$, and (ii) $K^c$ is the cost measured in terms of elapsed time associated with failure to load a page for class $c$. Some classes of clients may have more stringent requirements for response time for served requests; and the system has to guarantee low response time to avoid high value of $W^c$ associated with those classes. On the other hand, some classes may have low tolerance for page-view rejection; and the system will try to complete jobs without incurring their failure cost $K^c$.

The time cost for pageview each class is essentially summation from latency time cost of admitted pageview, $W^c \cdot (E(RT^c))$, and rejection time cost, $W^c \cdot (\phi^c \cdot K^c)$. The ultimate objective of this appendix is to build up a control mechanism minimizing the weighted cost of the system from classes, shown as following.

$$\min TC = \sum_c W^c \{(E(RT^c)) + K^c \cdot \phi^c\} \quad (A.10)$$

$E(RT^c)$ is the function of the connection time, server time and transfer time of all connections. From the equations A.4, we can clearly see that connection time can be controlled by arrival rate of SYN and time-out value of SYN retry. Server time depends on accepted SYN request and size of embedded objects and index.html. Transfer time can be managed by size of embedded objects and index.html. Nevertheless, page failure varies with number of rejected SYN ($\phi$) requests.

We, therefore, consider following three control knobs: (i) arrival rate of SYN ($\lambda^c_{i,j}$) requests, (ii)
time-out value of $j^{th}$ SYN retry($T_{O_{c_{j}}}$) requests and (iii) size of embedded objects $s_{i,k}$ to achieve our objective. The first challenge of our control problems stems from obtaining optimal values for control knobs to minimize the total cost. The second challenge lies on utilizing the system statistics provided by Ksniffer and use existing Ksniffer functionalities to tune the control knobs. Currently, Ksniffer is able to throttle and retransmit SYN requests to operate on first two control knobs and we would be dealing with these two only for our purpose.

A.3.1 Steady-State Control Optimization

Prior to solving equation A.10, we analyze the interactions of aforementioned three control variables. Size of embedded objects affects server time and transfer time of a page-view, especially because of network congestion. With the nature of the workload seen by most three-tiered web servers (described in section A.1), embedded objects are only served from the frontend Apache server, which is much under-utilized whereas backend database server is often fully utilized. The size of embedded objects accounts for very little portion of server time of a single-page, especially during the high traffic load. On the contrary, compared to index.html, embedded objects take much larger time and account for majority of transfer time. Therefore, we only consider to rewrite and remove the embedded objects’ size for effective control of transfer time of pageview. It also leads us to decouple the size of embedded objects (third control variable) (iii) from other two control variables when optimizing the objective function A.10. We first try to optimize the arrival rate of SYN ($\lambda_{i,j}^c$) and time-out of $j^{th}$ SYN retry($T_{O_{c_{j}}}$) requests with given values of embedded objects size and transfer time. Secondly, size of embedded objects $s_{i,k}$ will be optimized to manage the transfer time.

Due to the retransmission capability (by Ksniffer) on behalf of client, we can rewrite the $E(RT^c)$ shown in equation A.2 to optimize the objective function. Since Ksniffer can retransmit SYN on behalf of client, we can overlook the limitation of current implementation of TCP exponential retransmission policy by ignoring the second and third SYN retries. Consequently,
connection time of the original problem can be posed as function of accepted first SYN transmission arrival rate $\lambda_i$ of connection $i$ and plus holding time ($TO_i^c$) prior to entering the system. Therefore, $E(RT^c)$ and $\varphi^c$ can be simplified as following way:

$$E(RT^c) = \max_i \frac{\lambda_i^c}{\lambda_i^c + \lambda_i^{c,5}} \left( \sum_t \sum_k \theta_{(k,i)}^c \mu_t \left( \mu_t - \Lambda^t \right) + \sum_k \Gamma_k^c + TO_i^c \right)$$

$$\varphi^c = \max_i \left\{ 1 - \frac{\lambda_i^c}{\lambda_i^c + \lambda_i^{c,5}} \right\} \quad (A.11)$$

With the simplification, we arrive at the following lemmas:

**Lemma A.3.1.** Only accept first SYN and drop all the SYN retry.

**Proof.** Assuming $E(RT)^c$ as equation A.11, objective of equation A.10 is degenerated into a function with control variables of $\lambda_i^c$ and $TO_i^c$. For any given set of $\{\lambda_i^c\}$, $TO_i^c$ will be driven to 0 for minimizing the objective function. Therefore, holding time $TO_i^c$ prior to entering the system has to be zero for $i^{th}$ connections and $c$ classes for optimal set of $\{\lambda_i^c^*\}$ as well. This result implies that there exists an optimal arrival rate of $i^{th}$ of $c$ class, as $\{\lambda_i^c^*\}$, which is achieved by accepting those SYN at their first try. Assuming such policy is achievable, the second and third retry will be therefore dropped to avoid holding time.

**Lemma A.3.2.** Always accept the first SYN of consequent connections, if the first SYN of first connection is accepted.

**Proof.** With previous proposition, we know that structure of an optimal result is composed of only the arrival rate of first SYN on each connection; hence we can drop all subscript of $j$ for SYN retry. When the system have enough capacity to accept all the first SYN from all connections, the first SYN of all consequent connections can be accepted. When there is not sufficient capacity
(Cap) for SYNs from all connection, certain SYNs will be dropped. Here, we considers two types of dropping policy of class \(c\) corresponds to each connection \(\{(\varphi^1_i), \ldots (\varphi^2_i)\}\) and \(\{(\varphi^2_i), 0, \ldots 0\}\), where \((\varphi^1_i)^1 = 1 - \frac{\lambda^1_i}{\lambda^1_i + \lambda^2_i}\) and \((\varphi^2_i)^2 = 1 - \frac{\lambda^2_i}{\lambda^1_i + \lambda^2_i}\). Also, \(\sum_i (\lambda^c_i)^1 = (\lambda^1_i)^2 = \text{Cap}\). Since \(RT^c_i\) depends on the effective arrival rate \(\Lambda\), a function of \(\sum_i \lambda^c_i\), we can infer \(RT^c_i = RT^c_i\). It’s clear that \(\varphi^1 = \max \varphi^1_i \geq \varphi^2 = \varphi^2_i\). Hence, for any class \(c\), \(\{(\lambda^c_i), 0, \ldots 0\}\) can return the minimal failure cost for a given effective arrival rate for class \(c\). And there exists \(\{(\varphi^c_i)^*, 0, \ldots 0\}\forall c\), the optimal solution for equation A.10. This structure implies to reject certain SYN of first connections and accept the consequent connections with accepted first SYN, when there is limited capacity.

With Lemma A.3.1 and Lemma A.3.2, admission control to manage pageview response time is boiled down to first connection of each page. We, henceforward, (i) drop control variables \(TO_j\) and (ii) only consider arrival rate of first connection \((i = 1)\), and ignore other connections \((i > 1)\). Hence, the max operator can be removed from the original objective function and be re-written as following

\[
\text{(P2)} \min \sum_c W^c \{E(RT^c) + K^c \cdot \varphi^c\}, \text{where}
\]

\[
E(RT^c) = \left\{ \sum_k \Gamma^c_k + \sum_i \sum_k \theta^c_{(k,i)} \frac{\mu^1}{\mu^1 - \Lambda^1} \right\}
\]

\[
\varphi^c = 1 - \frac{\lambda^c}{\lambda^1_{1,0} + \lambda^c_{1,5}} \tag{A.12}
\]

As such, P2 can be solved with existing nonlinear optimization solver to obtain \(\{\lambda^1_1, \ldots \lambda^C_1\} = \{\lambda^*_1, \ldots \lambda^*_C\}\).
A.3.2 Transient SYN Management

In order to achieve \( \{\lambda^1, \ldots, \lambda^C\} \), Ksniffer can either throttle the SYN of first connection to the Ksniffer buffer and retransmit SYN from the buffer. Ksniffer can take the decisions of (a) SYN throttling and (b) SYN retransmission by actively checking if \( \{\lambda^1, \ldots, \lambda^C\} \) has been archived through system statistics. We consider observable arrival rate of all the classes to devise a transient SYN control mechanism at Ksniffer.

Note that the following proposed transient control mechanisms are to achieve lemma A.3.1, lemma A.3.2, and \( \{\lambda^1, \ldots, \lambda^C\} \) from (P2). Each of them accepts the first SYN of consequent connections for every pageview once their first connections are accepted. Therefore, pageview connections can be tracked through the first SYN of the first/index.html connection. Each of them further tries to control number of the first SYN of the first connection based on \( \{\lambda^1, \ldots, \lambda^C\} \). Once the first SYN is place the ksniffer buffer, it can be hold up to 21 seconds to facilitate tracking and throttling of the second and the third SYN. And the retransmission always start from the youngest one in the buffer stack for all three mechanisms.

A.3.2.1 Observed Inter-arrival Time

Inter-arrival time of class \( c \) (IAT\(_c\)) equals to \( \frac{1}{\lambda^c} \). Tracking IAT\(_c\) reflect whether \( \lambda^c \) is achieved. We define average inter-arrival time by the past \( F \) admitted first connection as following

\[
IAT_t^c = \frac{\text{(Time between the last one and last F admitted connections)}/(F)}.
\]

When a new SYN arrives and current \( IAT_t^c \leq \frac{1}{\lambda^c} \), it will be throttled; otherwise, it will be accepted. When a pageview is completed and current \( IAT_t^c \geq \frac{1}{\lambda^c} \), Ksniffer will retransmission SYN to increase \( \lambda \) from the youngest one to reduce the effect of waiting time on response time. Since IAT\(_c\) is only computed through the accepted SYN arrival, IAT\(_c\) may not get updated at page departure when arrival rate is higher than departure rate and new SYN just directly enters ksniffer buffer. As result, SYN in Ksniffer buffer won’t get chance to be retransmitted, even though there is service capacity for it. In order to prevent this deadlock scenario, we also use periodical IAT\(_c\) check to
For every SYN arrival,
   If it’s first SYN of second and consequent connections, accept.
   If it’s first SYN of first connection,
      If $IAT^c \leq \frac{1}{\lambda c}$, accept it and update $IAT_i$;
      otherwise, keep them in the Ksniffer buffer.
   If it’s second and third SYN, drop it.
For every first connection page departure,
   If $IAT_i \leq \frac{1}{\lambda c^*}$, retransmit youngest first SYN in the buffer.
   Update $IAT_i$.
For every $t$ seconds,
   If $IAT_i < IAT^*_i$, retransmit youngest first SYN in the buffer.
   Update $IAT_i$.

Figure A.7. Inter-arrival Time Control Algorithm

evoke retransmission mechanism.

A.4 Experiments Evaluation

We implemented NIAC as a set of kernel modules that can be loaded into an inexpensive, off-
the-shelf PC running Linux. Our kernel module approach is based on previous work which
demonstrated significant performance scalability benefits for executing within kernel space [59].
We present results using NIAC to manage weighted latency cost and failure cost in multiple
classes clients environments using TPC-W [89], a transactional web e-Commerce benchmark
which emulates an online book store, running on a three-tier web architecture. Figure A.8 shows
our experimental testbed, which consists of (i) web server machines, (ii) client machines and (iii)
TPC-W workload generator.

1. Three-tier web server machines

   We have three servers functioning as three-tier web server architecture. We used IBM
IntelliStation M Pro 6868 as Apache machine for first tier HTTP server. Apache 2.0.55 was installed and was configured to run up to 2000 server threads using the worker multi-processing module configuration. The Tomcat machine as the second tier application server (servlet engine) was an IBM IntelliStation M Pro 6849 with a 1.7GHz Pentium 4 CPU and 768MB RAM. Apache Tomcat 5.5.12 [90] was employed and configured to maintain a pool of 1500 to 2000 AJP 1.3 server threads to service requests from the HTTP server, and a pool of 1000 persistent JDBC connections to the database server. The third tier database (DB) server machine was an IBM IntelliStation 6850 with a 1.7GHz Xeon CPU and 768MB RAM. MySQL 1.3 was employed and set to the default configuration with the exception that the max_connections was changed to accommodate the 1000 persistent connections from Tomcat.

2. Client machines

We use three client machines of IBM IntelliStation M Pro 6868 with a 1GHz Pentium 3 CPU and 512MB RAM to function as three different classes, Gold, Silver and Bronze. Each of them use Java implementation of TPC-W [91] for workload generation of 450 clients with two modifications [58]. The TPC-W e-Commerce application consists of a set of 14 servlets to dynamically generate container page of each pageview within Tomcat. The servlet first performs a DB query to obtain a list of items from one or more DB tables, then the container page is dynamically built to contain that list of items as references to embedded images. After the container page is sent to the client, the client parses it to obtain the list of embedded images, which are then retrieved from Apache. As such, all images are served by the front end Apache server, and all container pages are served by Tomcat and MySQL.

Note that all machines were running RedHat Linux, with the DB server running a 2.6.8.1 Linux kernel and the other machines running a 2.4.20 Linux kernel. The machines were connected via 100Mbps Fast Ethernet Netgear, CentreCOM, and Dell switches. A modified version of the
rshaper [76] bandwidth shaping tool was installed on each of the three client machines to emulate wide-area network conditions in terms of transmission latencies and packet loss.

### A.4.1 Service Requirement Inference

In order to validate the proposed decomposed pageview model, we need to first to infer the service rate \( (\mu_{k,c,t}) \) of type \( k \) and class \( c \) at server tier \( t \). We investigate two methods for service requirement inference, open queueing network and close queueing network. Tool of Ambience developed by IBM TJ Watson is used for open queueing network and mean value analysis is adopted for close queueing network. Based on statistical profiling of pageview, we specifically consider three types of requests, container object, big embedded object, and small embedded object. Since only container objects need to be processed at all three items of server and Apache server is mostly unfertilized, we present model fitting for container object from a single class at Tomcat and Mysql server.

We calibrate open queueing model with different set of arrival rate by Ambience. With
inferred service requirement, model fitting results of responds time and utilization are shown in figure A.9 with actual measure data from the system. On the other hand, figure A.10 is generated with various input of number of active client population to calibrate response time and utilization for close queue model. We can see that both of methods show reasonable fitting, especially for utilization. Actually, service requirements of container object at Tomcat and Mysql from both methods are very close. Open and closed queueing network are equivalent in theory, so that both of methods eventually return similar service requirement inference. Depending on the availability of statistics collection and system tanning, both methods can effectively provide service requirement inference as modeling input for the environment of multi-tier server and multi-classes.

A.4.2 Validation of Decomposed Pageview Model

Prior to showing effectiveness of optimizations control design, we first demonstrate how a decomposed pageview model of a single class client predict client perceived pageview response time
with different clients population. Figure A.11 depicts comparison of model predicted pageview response time v.s observed one, which is decomposed into connection time, server time, and transfer time of index.html and embedded objects over two connections. Model takes observed arrival rate of SYN transmission at multiple connections to predict connection time. Transfer time of different type objects is modeled through average values, which remains more or less constant at various loads. Server time of container object and embedded objects can be obtained either through open or closed queuing model, described in previous subsection. If a pageview fails, it’s assumed to have 21 sec. of perceived pageview response time, associating with connection time component.

From figure A.11, decomposed model overall can provide response time prediction within 10% error. With increase of client population, percentage/weight of server time of html increases greatly, due to the saturation of Mysql. Meanwhile, percentage of connection also increases because of pageview failures. Comparing the results (not shown here) with 550 clients with (a) 1100 threads and (b)400 threads Apache threads, the percentage of connection and server time got swapped and total pageview response time remains the same for both of cases. Small values of Apache threads throttles out SYN and reduce the server load; html server time therefore decreases but connection time increases because of the second and the third SYN retransmission under TCP exponential mechanism. This phenomenon has also been discussed and observed in
Due to such a inter-correlation of connection time and server time, it adds another dimension and essentiality to model and control both of components simultaneously at high traffic rate.

A.4.3 Transient Control

We use the calibrated service rate ($\mu$) and estimated client arrival rate ($\lambda$) under 1350 clients of Gold, Silver and Bronze classes to optimize P(2) in subsection ?? by Gams, an optimization software. Here, the following transient control results is based on the cost per unit time, $W^c$, for each class set to be [100, 30, 15] and perceived failure time cost, $K^c$, as [40, 21, ]. As mentioned earlier, Gold class has higher response time requirement for admitted pageview and also place emphasis on perceived failure pageview response time. On the other hand, Silver class only stresses on benefit of admitted requests and failure pageview incurs higher client perceived time for Bronze class. Based on that, class-wise optimal arrival rate, $\lambda^c\ast$, is calculated and translated into the optimal threshold values for inter-arrival time, concurrent connection, and product of arrival rate and admitted response time in the following subsections.

A.4.3.1 Interarrival Time Control

Figure A.12 shows total cost surface generated by analytical surface with admitted Gold average arrival rate for x-axis, admitted Bronze average arrival rate for y-axis, and accept all pageview requests from silver class. It’s a convex shape surface, which decreases with Gold class for given admitted Bronze class. From the graph, we can see minimal values of TC lies at $[IAT^G, IAT^B]=[1/14,1/14]$, which coincides with optimization result returned from Gams. We have conducted extensive experiments with different IAT threshold as extra data points on the top of surface to verify the analytical surface and effectiveness of IAT control to achieve the minimal TC.

Extra dot points were generated with fixed silver class threshold, and thresholds for Bronze and Gold are as $\frac{1}{x+value}, \frac{1}{y-value}$; each of data points takes 30 minutes to generate. From figure
A.12, we can see that such a scheme can effectively achieve corresponding desired arrival rate by taking observed inter-arrival for guideline of SYN throttling and retransmission. As for TC values, such points match well with analytical surface and show effective predictability of proposed models. Also, optimal result returned from GAM also suggests the same minimum point as the analytical surface.

A.5 Conclusion

The proposed Non-intrusive Admission Control (NIAC) successfully manages weighted cost (obtained from client perceived admitted pageview response time and perceived failed pageview response time) for multiple classes of clients. Earlier studies have mainly focussed on this problem by considering a component of the total perceived response time (such as server end pageview response time) and hence, can be described as “incomplete” at their best. The proposed NIAC-based approach offer (i) detailed decomposed pageview response time model, (ii)multi-class steady-state optimization, and (ii)transient inter-arrival rate admission control. Without modifying the multi-tiered architecture of the application, NIAC infers system parameters leveraging two types of modeling techniques, namely, open and close network. Optimal multi-classes’ arrival rates from steady state optimization based on decomposed pageview model can be well achieved.
by proposed transient inter-arrival rate admission control. Overall, NIAC comprehensively models actual client perceived pageview response time and provides simple implementable admission control policy from observed inter-arrival rate to achieve desirable results. The effective transient admission control results using NIAC on client perceived response time leads this appendix to investigate more observable system parameters, such as queued length and active concurrent connections, for future work.


Vita

Yiyu Chen

Education

• Ph.D in Industrial Engineering and Operations Research with Minor in Statistics 2002-present
  The Pennsylvania State University, University Park, PA

• B.A. in Economics and Industrial Management  June’02
  National Taiwan University, Taipei, Taiwan

• Exchange Student  Sep’00-June’01
  British Columbia University, Vancouver, Canada

Publications


Patent Application