NOTE TO USERS

This reproduction is the best copy available.

UMI®
Resource Management in Large-Scale Services:
Models and Algorithms

Daniel Antunes Maciel Villela

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts and Sciences

Columbia University

2005
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
ABSTRACT

Resource Management in Large-Scale Services:
Models and Algorithms
Daniel Antunes Maciel Villela

Many of the services in today’s Internet (e.g. file transfer, streaming media, auctions) are offered by companies that either own or lease a fixed, static set of server resources to host these services. Under such a structure, a company can provision their serving resources for its typical loads or to conservatively handle peak demands. Either way has drawbacks. In the former, poor performance is expected when peaks of demand occur. In the latter, costly resources remain idle most of the time.

This thesis explores provisioning alternatives that use pools of servers in novel ways that permit provisioning for typical loads but can still accommodate peak demands. Three specific problems are explored by constructing mathematical models and analyzing these models.

First, we explore benefits for independent companies from pooling together of their own servers to jointly serve data delivery. This data may have real-time requirements (e.g., multimedia), or it may be elastic (e.g., file transfer). We show that such pooling can simultaneously benefit all companies participating in this arrangement, but that it is necessary for these companies to have similar demands and to offer similar quantities of serving resources.

Second, we explore how to reduce transaction times for transaction-oriented services, such as an auction website. If such a service is replicated across multiple servers, these servers must be kept consistent, and replicating the service across many servers will do more harm than good, as the cost to maintain consistency will offset the gains made from
distributing the load. We explore the performance of heuristic-based algorithms that identify provisionings that are close to optimal.

Last, we consider a general provisioning problem for a business that owns a server farm, and must decide how to allocate its fixed pool of servers among a set of independent companies that offer transaction-oriented services. The optimal allocation is a function of the loads imposed by each company, the profit each company offers for a successful transaction, and the cost that the server farm must pay for each failed transaction. We derive three approximation methods for partitioning the fixed set of servers among the various companies.
Contents

Contents i

List of Figures v

List of Tables viii

1 Introduction 1

1.1 Overview .................................................. 1

1.2 Alternatives for Provisioning of Service Resources .............. 3

1.2.1 Load Sharing vs. Load Balancing .......................... 4

1.3 Challenges .................................................. 5

1.3.1 Resource pooling and Competing Objectives ................. 5

1.3.2 Services with Consistency Requirements ..................... 7

1.3.3 Profit for a Third-party Server Provider .................... 9

1.4 Contributions of this Dissertation ................................ 11

1.5 Structure of this Dissertation .................................. 12

2 Load Sharing in Content Distribution Services 14

2.1 Overview .................................................. 14

2.2 Related work ................................................. 17

2.3 Collectives General Model .................................... 19
2.4 Collectives under Fixed-rate Transfers ........................................ 22
  2.4.1 Computing the rejection rate of a collective .......................... 23
  2.4.2 Evaluation of Scenarios under Fixed-rate Transfers ............... 25
  2.4.3 Asymptotic limits of collectives: The $\rho/k$ factor ............... 27
  2.4.4 Win-only areas and servers' capacities .............................. 29
2.5 Collectives for Elastic Transfers ............................................. 31
  2.5.1 Numerical evaluation ..................................................... 34
2.6 Resource Bounding with Thresholds ......................................... 36
  2.6.1 Analytical evaluation of D1 thresholding ............................ 37
  2.6.2 Comparison of Thresholding Techniques ............................. 40
  2.6.3 Extending Heterogeneity .................................................. 43
  2.6.4 Optimal Thresholding ...................................................... 45
2.7 Experimentation of Collectives for a Video Delivery Service: proof of concept ................................................................. 47
  2.7.1 Thresholding ................................................................. 48
  2.7.2 Implementation and Experiments ....................................... 49
  2.7.3 Methodology ................................................................. 50
  2.7.4 Results ...................................................................... 53
2.8 Conclusion .................................................................................. 54

3 Load Sharing in Services with Consistency Requirements .................. 57
  3.1 Overview ............................................................................. 57
  3.2 Related Work ....................................................................... 60
  3.3 Large-Scale Services with Consistency Requirements: model and objectives ................................................................. 63
    3.3.1 Modeling a servicing system ............................................ 64
    3.3.2 Objective: maximum intensity ........................................ 67
### 3.4 Algorithms for load–sharing

- 3.4.1 Useful definitions and facts ........................................... 70
- 3.4.2 Description of the proposed greedy algorithms ................. 72
- 3.4.3 Analysis of GREEDY (k) ............................................. 76
- 3.4.4 Analysis of GREEDY–FLUID ....................................... 80
- 3.4.5 Quantitative analysis .................................................. 82

### 3.5 Fast fluctuation of demands ............................................ 91

- 3.5.1 Extension of model ..................................................... 93
- 3.5.2 Analysis of the intuitive case $z = 2$ ............................. 94
- 3.5.3 Generalized Analysis .................................................. 95
- 3.5.4 Numerical Results ..................................................... 97
- 3.5.5 Discussion ............................................................... 99

### 3.6 Conclusion ............................................................... 100

### 4 Third–party Resource Management of E–Commerce Websites  

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Overview</td>
<td>103</td>
</tr>
<tr>
<td>4.2</td>
<td>Related Work</td>
<td>106</td>
</tr>
<tr>
<td>4.3</td>
<td>Model for the Application Serving System</td>
<td>108</td>
</tr>
<tr>
<td>4.4</td>
<td>Application–tier Workload Characterization</td>
<td>112</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Methodology</td>
<td>113</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Trace Analysis</td>
<td>114</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Analysis– why Poisson arrivals</td>
<td>121</td>
</tr>
<tr>
<td>4.5</td>
<td>Server Provisioning</td>
<td>123</td>
</tr>
<tr>
<td>4.5.1</td>
<td>General Solution Procedure</td>
<td>123</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Costs as function of average response times</td>
<td>126</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Costs as function of variance of response times</td>
<td>127</td>
</tr>
<tr>
<td>4.5.4</td>
<td>The EBB–based Method</td>
<td>129</td>
</tr>
</tbody>
</table>
4.5.5 Solving allocations via comparative and first-allocation equations 133

4.6 Experiments .................................................. 134
    4.6.1 Experimental Results ................................ 136
4.7 Determining an optimal number of servers to deploy .......... 140
4.8 Conclusion ................................................... 141

5 Summary and Future Work ..................................... 143
    5.1 Summary of the Dissertation ............................. 143
    5.2 Future Research Directions ............................... 148

Bibliography ......................................................... 151

A Rejection Rate in Processor Sharing Model .................. 168

B Outline on the implementation of the protocol of SERES platform 171
    B.1 Adherence to SERES ..................................... 172
    B.2 Content Exchange ....................................... 172
    B.3 Redirection Inquiries .................................... 173
    B.4 SERES Table ............................................. 174
    B.5 Examples ................................................ 174
List of Figures

1.1 The typical use of resources to serve a transaction from a client to a server. 1

2.1 Collective example compared to systems in isolation. .......................... 20
2.2 Evaluation of collectives under the model of fixed-rate transfers. ............... 25
2.3 Asymptotic observations on rejection rate for fixed-rate transfer collectives. 28
2.4 Evaluation of collectives under the model of fixed-rate transfers by varying values of capacities $k_1$ and $k_2$. .................................................. 30
2.5 Distinct configurations by varying $k_1$. ............................................. 31
2.6 Distinct configurations at normalized values by varying $k_1$. ................. 32
2.7 Evaluation of a collective for elastic transfers. ..................................... 34
2.8 Areas of benefit for collectives under processor sharing discipline .......... 35
2.9 Respective rejection rate experienced on requests for providers $y_1$'s and $y_2$'s content and different thresholds. .............................. 39
2.10 The use of D1–thresholding and D2–thresholding/switching. .................. 40
2.11 Comparison of D1–thresholding and D3–thresholding/switching. .............. 42
2.12 Extending heterogeneity using thresholds. ......................................... 44
2.13 Optimal thresholding. ................................................................. 45
2.14 Bandwidth consumed by servers under SERES. .................................. 52

3.1 GREEDY (k) algorithm ................................................................. 73
3.2 GREEDY-FLUID algorithm

3.3 Configurations obtained after movements given by the even-size fragmentation model (center) and the fluid fragmentation model (right). By contrast, a configuration without load sharing is shown in the left.

3.4 Results obtained using GREEDY(k) and GREEDY-FLUID for gaussian, fast-decaying, and linear inputs, \( f_i = .1, 1 \leq i \leq m = 16 \). Note: a re-labeling of servers is done during both procedures.

3.5 Results obtained using GREEDY(k) and GREEDY-FLUID for gaussian, fast-decaying, and linear inputs (all randomized), \( f_i = .1, 1 \leq i \leq m = 16 \). Note: a re-labeling of servers is done during both procedures.

3.6 Results obtained using GREEDY(k) and GREEDY-FLUID for fast-decaying input function, \( f = .2 \) in (a), (b), (c), \( f = .3 \) in (d), (e), (f), \( m = 16 \). Note: a re-labeling of servers is done during both procedures.

3.7 Results obtained using GREEDY(k) and GREEDY-FLUID for linear input, \( f \) described by a linear, decreasing function, \( m = 16 \). Note: a re-labeling of servers is done during both procedures.

3.8 Results obtained using GREEDY(k) and GREEDY-FLUID for gaussian input, write intensities \( f_i \gamma_i \) equal for any \( a_i \in \mathcal{A}, m = 16 \). Note: a re-labeling of servers is done during both procedures.

3.9 Results obtained using GREEDY(k) and GREEDY-FLUID for fast decaying input, \( f_i = .1 \) for any \( a_i \in \mathcal{A}, m = 1000 \). Note: a re-labeling of servers is done during both procedures.

3.10 Results obtained using GREEDY(k) and GREEDY-FLUID for gaussian input, \( f_i = .1 \) for any \( a_i \in \mathcal{A}, m = 1000 \). Note: a re-labeling of servers is done during both procedures.
3.11 The average between asynchronous periods of high and low intensities
for two distinct servers containing two distinct instances each. 92

3.12 The case of two servers and two instances. Here, $\omega = 1$ in high periods,
$\omega' = .1$ in low periods, $c = 2.5$, $f = .111$ (in Figure 3.12(b)), and $p = .2$
(in Figure 3.12(a)). 97

3.13 The case of many instances but still a homogeneous system. ($m = 60$,
$p = .1$, $c = 3$, $f = .25$, $\omega(a) = 1$ in high periods, $\omega'(a) = .01$ in low
periods). 98

4.1 IDI values and load over a 24-hour period ($k = 20$). 114

4.2 The IDI test over different timescales using traces of a dept. store website. 115

4.3 Characterization of workload including requests for not only dynamic
content but also static content. 117

4.4 IDI values for different traces. 118

4.5 Distribution of length of sum of $k$ consecutive intervals. 120

4.6 Different configurations under different provisioning schemes. 137

4.7 Costs incurred when varying only one customer (1) arrival rate ($\Lambda_1$). 137

4.8 Costs obtained when varying the number of customers. 139

B.1 Server $S_2$'s adherence to a collective. 175

B.2 Server $S_1$ inquiry $S_2$ to deliver a session for client $C$. 175
List of Tables

2.1 Terminology, main variables, and main parameters of the model ........ 21
2.2 Number of occurrences of events and rejection rate during experiments .. 52
3.1 Main variables and main parameters of the model ..................... 64
Acknowledgements

I was fortunate to have help and support from numerous people during my time in the PhD program at Columbia University. Here, I have the opportunity to thank them.

First, I would like to thank my thesis advisor, Dan Rubenstein, for supporting me in the program and encouraging me to pursue my own research interests.

Over the course of the program at Columbia and during my summer internships at AT&T Research in 2001 and IBM Watson Research in 2003 I met researchers, professors, colleagues and friends with a great sense of how to do research, which helped me to develop my own view of research. In particular, I have special mentions for Vishal Misra, Prashant Pradhan, Sambit Sahu, John Vicente, Michael Kounavis with whom I had the opportunity to collaborate, for Andrew Campbell who supervised my work in my first two years in the doctoral program, and for Otto Carlos Duarte who supervised my masters thesis work in Brazil (prior to coming to Columbia) and encouraged me to advance to a PhD program. I also especially remember folks from the Comet lab, DNA lab, EE department, and CS department. I would also like to thank Wei Ke for helping in the implementation of a client program for experiments with a streaming server.

It is also important to mention that in my first four years in the program I received scholarship support from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq–Brazil), an agency of the Brazilian federal government, under scholarship ref. number 200168/1998-3.

I would also like to thank Edward G. Coffman, Jr., Predrag Jelenkovic, Jason Nieh and Prashant Pradhan for serving on my thesis defense committee.

I am grateful to my parents and family on both sides who have always been very supportive to me. My dear wife Sylvia has always given me enormous support, which definitely strengthened my motivation for doing this work. I dedicate this thesis to her and to our children, Adrian and Juliana, who were born during my PhD journey.
To Sylvia and my children, Adrian and Juliana
Chapter 1

Introduction

1.1 Overview

Numerous websites and applications in networked environments such as the Internet are regularly accessed by large numbers of users. Due to their large scale, the management of resources serving such applications poses various challenges. A user access is usually characterized by a process that starts when the user issues a request to the website, e.g. hitting a webpage "link", via a client application, typically a browser. The request is directed over a network path to a machine capable of processing requests. In fact, the machine is usually reserved for processing requests and, for this special purpose, is termed a server. After processing the request, the server sends a response back to the user.

Figure 1.1: The typical use of resources to serve a transaction from a client to a server.
Figure 1.1 illustrates a client, a network path and a server in a rounded-corner rectangle, a thick line, and a circle, respectively. In this figure we only illustrate the case of a single client. In the actual scenario of multiple clients, multiple requests co-exist in portions of the network path, typically across the access link shown in the figure, and are also processed simultaneously in the server. The number of resources placed on the network path or at the server can affect performance metrics, such as the time needed to complete the transaction. It is common to refer to the limiting resource as the system bottleneck. In this thesis, we address fundamental problems related to provisioning of resources for various kinds of services including content distribution, online auctioning, and e-commerce, which are offered in networked environments such as the Internet. The providers of these services must provision an infrastructure of computing and networking resources such as servers and network links in order to handle demands on large-scale.

We consider two possible bottlenecks in such serving systems. The providers' servers can be the bottleneck in the face of high demands, due to the large amount of processing required. This happens because a server's capacity, defined by the amount of work it produces at a unit time, is finite. For instance, particular kinds of servers serving e-commerce websites (application servers) execute scripts that may be time-consuming. The server is then a bottleneck, since it has to process a large number of processes simultaneously. The access link in the network path can also be a bottleneck. In particular, the bandwidth available in the access link (Figure 1.1) by the provider [83] can be the bottleneck, since network links also have finite bandwidth.

Later in this dissertation, we describe how, in some cases such as content distribution services, the limiting resource at the bottleneck can either be access bandwidth or server processing. In other cases, such as online auctions, the resource to be provisioned is mainly that of server processing. We construct models for each case to describe the fundamental aspects, which can then be any of the resources identified as possible bottle-
necks. In this chapter, however, for the sake of coherence we only describe servers as the resource to be provisioned.

1.2 Alternatives for Provisioning of Service Resources

The solution of using a single server for a large-scale service should be avoided since it does not scale to large demands. Besides, even if a very powerful server can handle the demand of a service, the financial cost of purchasing a very powerful server and maintaining it can be substantially higher than having a set of cheaper servers [15].

Instead, provisioning resources for a service may require the provider to first make a number of decisions. First, a provider must dedicate resources to its serving infrastructure. Financial factors, such as the cost of purchasing, setting up, and maintaining servers, limit the amount of resources that the provider can utilize. Second, providers should "engineer" the use of the resources to which they have access, i.e., use them in a way to optimize a desired metric, such as response time. In essence, providers should distribute servicing tasks across the serving infrastructure to meet performance guarantees.

How to distribute servicing tasks across resources becomes more challenging to when multiple resources are involved.

Let us consider a "toy" example of two servers, X and Y, that deliver streaming media content A and B, respectively, and that each server accepts a maximum of 10 streaming sessions (i.e., a static provisioning) due to a constraint on access bandwidth. If, in contrast, the two servers were each capable of delivering both content A and B, the maximum number of simultaneous streaming sessions for either piece of content increases to 20. The probability of reaching the maximum number of simultaneous sessions may diminish. As a consequence, using this more flexible arrangement service is denied less often than in a static provisioning arrangement.

A preliminary solution, as initially suggested in the previous example, is to partition
the set of all possible tasks of the service into subsets such that tasks in each of these subsets are only served by a single server. Many services exhibit a demand that is an aggregate of demands, each due to a servicing instance, defined as all tasks necessary to serve a common object. A natural way to partition the tasks is according to the servicing instance, such that all tasks for a same servicing instance are served by a single server.

For example, for the delivery of a set of videos, tasks could include the retrieval of video, pause, advance play back and so on. All tasks possible for servicing a same object—in this case a video—compose a single servicing instance. Each video (or a subset of all videos) can be placed on a single server to which streaming requests (for playback, pause, advance) are directed. We refer to this provisioning alternative as static provisioning.

In this thesis, we study alternative provisioning solutions such as the one described in the previous example. These alternatives engineer the servicing across the serving infrastructure for load sharing, i.e. such that tasks can be served at a choice of multiple servers, and dynamic re-provisioning. Servers can then be dynamically designated for servicing subsets of servicing tasks. The provisioning via these alternatives requires solving challenging problems, which we describe at a high level in this chapter and in detail in the next chapters. We demonstrate, however, that such alternatives can significantly improve the performance of these services.

1.2.1 Load Sharing vs. Load Balancing

Before describing the research problems, it is useful to explain the fundamental difference between the terms “load sharing” and “load balancing”. The former refers to a policy in which the goal is to have multiple resources “share” the servicing of the workload. The latter refers to a policy whose goal is to maintain load at servers at equal levels. Load sharing applies more broadly to reach a particular objective such as minimum response

---

1.1 This is in contrast to executing a single task with huge processing in parallel over multiple servers.
times, regardless of the load levels. In one of the earliest analytical studies [39] on load sharing it is stated that:

"We consciously use the phrase load sharing rather than the more common load balancing to highlight the fact that load balancing, with its implication of attempting to equalize queue lengths system-wide, is not an appropriate objective."

We show that the provisioning of a service applying load sharing can be useful, but we also identify challenges that make its application non-trivial to be performed.

We describe next the challenges that more elaborate provisioning methods present in modern large-scale services.

1.3 Challenges

1.3.1 Resource pooling and Competing Objectives

In the toy example that we described previously, when A is highly overloaded and B is underloaded, A may start to reject requests. A solution for reducing A's lack of "responsiveness" (measured by the rejection rate) that utilizes load sharing transfers servicing from A's server to B's server. The obvious problem is that B may not be satisfied with this arrangement, although at subsequent times, the roles can be reversed and B would benefit from the arrangement, while A would not. On a large time scale, resource sharing can benefit providers of multiple services but provisioning is necessary, since competing objectives might be an adverse factor.

It is fair to consider that, given the stochastic nature of an actual system, if both A's and B's demands are close to one another, they may both indeed benefit from performing load sharing. This win-win situation motivates us to formulate an analytical framework that
can describe the conditions under which multiple providers benefit from pooling together resources.

The work here is motivated by popular content distribution services, such as delivery of streaming media. Content providers profit from servicing their clients’ requests for their content. If the processing capability of a provider’s servers is insufficient, a large number of requests during periods when the content reaches its peak in popularity are turned away. The amount of serving resources needed during a peak period, however, is often much larger than what is needed on a regular basis. Hence, a provider that provisions resources for these peak periods will pay for equipment that sits by idly most of the time, reducing profits.

Resource pooling among multiple providers can result in less frequently turning away requests. For instance, load can be balanced among the pooled resources. As a result, overloads (dropping requests) are less likely to occur [39, 38, 34, 112, 111]. There are, however, limitations on resource pooling since multiple providers each have their own objectives, which is to provide the best delivery of its own content. This is viewed clearly in the example described above. This approach permits providers to provision their serving infrastructure as a function of average demands, as an alternative to overprovisioning or contracting a content distribution network service [3]. We term a collective to be the structure formed by such mutual agreements to form a pool of servers.

The competing aspect is the factor that makes this problem distinct from ones studied in previous work: each service provider “profits” only from requests for its own content. While the collective more efficiently serves the aggregate demand (over all providers), because a provider’s resources may be used to help serve other providers’ content, there may not be enough resources in the collective to serve its own client demand. This leaves open the possibility that the rejection rate of requests for an individual provider’s content will be higher within the collective than if that provider operated in isolation. Since the
provider profits only from requests for its originating content, this increase in rejection rate can deter its participation in the collective.

Therefore, we aim to investigate the conditions under which all provider participants in a collective benefit from their membership in the collective. We also pursue mechanisms that can be used to increase the range of conditions under which all participating providers benefit.

1.3.2 Services with Consistency Requirements

We next turn our attention to systems with consistency requirements, since consistency is a factor present in many of today's services such as online auctioning websites. For the provider of such services it is important to deliver fast responses to its users.

Let us consider an example of service that clearly illustrates the consistency issue: an online auctioning service. Let a provider service auctions $a$ and $b$, respectively. The goal of the provider is to respond quickly to user requests, regardless of demand. We observe a scenario different from the content distribution example, because there is only one provider here, hence no competition can exist among different parties. Response times can be reduced by reducing the number of requests served at servers. Let us initially consider that the provider uses a single server for each of the auctions. If each of the auctions require the provider to respond to $\gamma$ requests per minute, then cutting the request rate by half at each server, but adding another half for the other auction on that same server, results in an equal request rate $\gamma$ placed on each server. The rate is actually even higher if the service in question requires consistency. Some requests, such a successful bid, modify the data at the server, e.g. minimum bid. When the auction is served at both servers, data must be consistent across servers. Hence, for each arriving data update, the processing is doubled, one at each of the servers. Therefore the total number of requests increases, and now a server has to cope with updates for both auctions. It is clear that if the
reduction in the number of requests is larger than the number of updates to be added from the auction to be added to that server, such an arrangement should be used. Otherwise, the arrangement clearly should not hold.

We can generalize such insight to actual systems, whose total demand is comprised of an aggregate of demands for multiple servicing instances, e.g. the example of online auctions. The simplest approach here is to distribute requests for servicing instances such that instances are each served by a single server. Instance demands, however, fluctuate over time, thus, at moments of peak demands, response times increase at the more heavily loaded servers.

Less intense demands are placed on servers by applying load sharing, such that requests for any highly-demanding instance can be distributed among multiple servers. As a result, the highly-demanding instances are on average served faster. An instance's data, however, must be updated across all replications in order to preserve consistency, when multiple servers service that instance. If one replicates instances over an arbitrarily large number of servers the cost of maintaining consistency can outweigh the savings gained from distributing the load of requests. The key solution is to find the replication factor for each of the instances, that results in lightly-loaded servers.

It is our goal to study the impact of load sharing on provisioning of services with consistency requirements. In order to do so we investigate algorithms for distributing an instance's demand to be served over a set of servers. We can then compare the performance obtained in the configurations obtained as outcome of such algorithms to the performance of configurations that do not apply load-sharing.

---

1,2 In actual servicing systems, load sharing is indeed applied to these server infrastructures, although its application appears to be ad hoc. Hence, it remains an open question as to how one should assign servicing instances to servers to reduce response times.
1.3.3 Profit for a Third-party Server Provider

We, next, consider the business of a server provider that owns and maintains servers deployed to service distinct e-commerce websites. Here the idea is for a profitable business that provisions resources for a number of servicing instances, i.e. each e-commerce website. In this case we can view the server provider as a third-party entity that owns and maintains a pool of servers. This scenario, however, differs from the one in Section 1.3.1 since it was previously considered that a number of providers can pool together their resources, whereas here the server provider owns the servers. Each provider of an e-commerce website can contract with the server provider for servers. The providers then delegate the provisioning task to the server provider that in return has to meet performance guarantees (service level agreement, or SLA for short) as specified in the contract. SLAs are customarily defined via statements regarding response times, e.g. "do not exceed 5 seconds". The server provider then has to service a large-scale demand of servicing instances. We study how the server provider can provision its resources among its set of customers in such a way that its profits are maximized.

For example, suppose a server provider has 10 servers and provisions servers to two distinct websites A and B. Website A receives requests at a rate of 200 per minute, whereas website B receives requests at a rate of 100 per minute. The provider intuitively should deploy more servers to website A than to website B. But this might not be true for various reasons. First, it might not be true if the demand is priced higher for website B than to website A, say for every request served for website B and every two requests of website A the server provider has equal revenues. Second, it might not be true either if website B is less tolerant with delays. For instance, A might accept that a response is sent back within 5 seconds of processing time, whereas B accepts a response sent back only within 2 seconds.

The previous example can be extended to a scenario of multiple customers. To provi-
sion servers, variables such as arrival rates for each website and revenues contracted from each of the websites, level of delay tolerance, and possibly more, have to be taken into account. Additional challenges arise from other facts. First, the stochastic nature of request arrivals and service times makes it impossible for the provider to meet the conditions of the SLA for every request it hosts. Hence, we assume that, as part of the SLA, the provider is charged for each service miss: a request whose service does not meet the requirements specified in the SLA. It is important to reduce or eliminate service misses, since profit varies linearly with demands accrued from charges for service misses. Second, response times do not vary linearly with demands placed on servers. Hence, we need to develop a model that captures the response time as a function of the demands placed on a server. This is a fundamental building block for the provisioning of servers in such a scenario. Third, each of the server–contracting websites can exhibit fluctuations in demand. A solution for the resource management problem here is to perform dynamic re–provisioning, i.e. provision at intervals of time as a function of measurements of demands.

An approach for static provisioning is to have each instance (in this case, an e–commerce website) always serviced by the same server. The limitation is obvious, once known that demands fluctuate. There will be more service misses for a website whose demand increases, consequently reducing the profits. By contrast, dynamic re–provisioning permits the server provider to be constantly re–configuring the server infrastructure to the various e–commerce sites in order to keep its profits at maximum. Since provisioning should take into account as variables all demands to be estimated in this problem, a server allocation that maximizes profits is challenging.

We want to devise methods for server allocation under these conditions. The problem of server provisioning then requires a model amenable to permit dynamic re–provisioning with little complexity. However, it also requires a model reasonably complex to describe response times effectively in order to meet SLAs and ultimately to maximize profits.
1.4 Contributions of this Dissertation

We present here a summary of the contributions of this dissertation:

- We demonstrate that when providers have equal average demands, in the case of multiple independent content distribution providers with competing objectives, rejection rates are significantly reduced, but slight deviations that lead to a more heterogeneous set of demands separate providers on whether or not to form a pool of resources.

- We further demonstrate that simple thresholding methods extend the heterogeneity that can be tolerated such that all providers agree on forming a pool of resources. We experiment with three possible methods in three sets of simulation.

- The results obtained in the work on collectives are validated by a proof-of-concept implementation of a video delivery application.

- For systems with consistency requirements, we devise simple algorithms based on heuristics to produce load-sharing configurations that are shown to reduce response times significantly, compared to configurations that do not apply load-sharing.

- If, in static provisioning of services with consistency requirements, all servers are equally loaded, we demonstrate, that, counter to intuition, the average response times can be reduced.

- We devise simple methods that can be used by a third-party server provider to allocate servers to multiple e-commerce websites with an objective of maximizing profits. We demonstrate via simulations that the results obtained from our methods are better than those obtained from naive heuristics and are close to optimal.

- We demonstrate via trace analysis that an application server of e-commerce websites can be described as an $M/G/1/PS$ queueing system.
• In particular circumstances, the allocation of servers to be performed by the server provider is defined in a single closed formula. This is important for permitting fast dynamic re-provisioning solutions.

1.5 Structure of this Dissertation

The remainder of this dissertation is organized as follows.

In Chapter 2 we develop a model for describing competing objectives of content providers. This model is extensively analyzed to identify conditions for resource sharing as a function of the heterogeneity of the system and propose thresholding mechanisms. We finally describe a system implemented over the Darwin Streaming Media service [10] and experimentation that works as a proof-of-concept. The work described in this chapter appeared in [107, 108] in preliminary forms.

In Chapter 3 we develop models that capture the role played by consistency in services such as online auctions. The consistency factor is shown to make the problem complex (in the NP-complete class of problems). We describe simple algorithms that provide results close to the minimum of the largest of intensities of servers across a complete set of servers that forms the infrastructure. We also develop an extension for the model that describes the effect when fluctuation of demands occur within measurement intervals. The work described in this chapter appeared in [109] in a preliminary form.

In Chapter 4 we develop a model that describes SLAs and a profit-based objective for the provider of servers to e-commerce websites. The problem is posed as an optimization problem. We first study the workload of an application tier of e-commerce websites to effectively characterize the input of demands that such tier receives from various e-commerce websites. We develop three simple mechanisms that can be implemented to allocate servers to distinct e-commerce websites. We also experiment with simulations the validity of such mechanisms. The work described in this chapter appeared in [110] in
a preliminary form.

Finally we conclude in Chapter 5 with a summary of the work and a description of possibilities for future research in the area.
Chapter 2

Load Sharing in Content Distribution Services

In this part of the thesis we examine for serving of elastic traffic (e.g. static web content) or traffic with multimedia requirements (e.g. streaming media) how servers forming server collectives can reduce rejection rates. The objective of reducing rates of request rejections arises since high fluctuation of demands makes the provisioning of this kind of service more difficult.

2.1 Overview

Content providers profit from servicing their clients' requests for their content. If a provider's serving resources (e.g. servers and bandwidth) are insufficient, it will be forced to turn away a large number of requests during periods when the content reaches its peak in popularity. The amount of serving resources needed during a peak period, however, is often much larger than what would be needed on a regular basis. Hence, a provider that provisions resources for these peak periods will pay for equipment that sits by idly most of the time, reducing profits.

A recent solution used by many providers has been to contract third party content distribution networks (CDN) that host and service their content during peak periods. The provider, however, pays the CDN for its assistance, which again can reduce profits. In-
stead of relying on CDNs during peak periods, an overlooked alternative is for groups of providers to form *collectives* and host one another’s content. When the demand for content that originated at provider A peaks, exceeding its own serving abilities, it can redirect requests to other members of the collective whose available serving resources can handle these requests. In return, when the demand for content that originates at some other provider peaks, provider A’s serving resources can be used to help serve this other content.

It is well known that systems that pool together resources can outperform the performance of their individual components. For instance, load can more easily be balanced among the pooled resources, and overloads (dropping requests) are less likely to occur [39, 38, 34, 112, 111]. The problem we consider here, however, has an important distinction from these traditional works and from the CDN model: *each service provider “profits” only from requests for its own content.* While the collective more efficiently serves the aggregate demand (over all providers), because a provider’s resources may be used to help serve other providers’ content, there may not be enough resources in the collective to serve its own client demand. This leaves open the possibility that the rejection rate of requests for an individual provider’s content can be higher within the collective than if that provider operated in isolation. Since the provider profits only from requests for its originating content, this increase in rejection rate can deter its participation in the collective. The Content Distribution Internetworking (CDI) charter [35] at the IETF describes similar concepts and requirements for interconnection of content networks to collectives. The CDI model, however, lacks a performance analysis of the benefits of such a system.

Here, we identify from a performance perspective when these collectives are a viable alternative. In particular, we address the following questions:

- Under what conditions do all provider participants in a collective benefit from their membership in the collective?
• Are there any mechanisms that can be introduced into the collective architecture that will increase the range of conditions under which all participating providers benefit?

To enable us to focus on the performance aspects of this question, we start at the point where a set of providers have agreed to form a collective, have made copies of one another's content, and can redirect requests for a particular content object to any server within the collective with sufficient available capacity. When no server has available capacity, the request is dropped.

For each provider in the collective, we compare the rejection rate for its content (that it originated) when served within the collective to when it serves its content in isolation. Each provider is described in terms of its capacity (number of jobs it can serve simultaneously) and its intensity (the rate of requests for its content divided by the rate at which it serves requests). We find that collectives reduce rejection rates of all provider participants by several orders of magnitude when the collective is formed from a homogeneous set of providers with identical capacities and intensities. However, even slight variations in intensity among providers yield heterogeneous collectives in which the lower intensity participants achieve significantly lower rejection rates in isolation than within the collective.

We next consider whether all providers' needs can be met by growing the size of the collective, i.e., can the rejection rate be brought arbitrarily close to zero by simply increasing the membership to the collective? We identify a simple rule that is a function of the average intensity and the average capacity that determines whether the rejection rate converges to zero or to a positive constant. A convergence to zero implies that all providers would benefit from participating in very large collectives. However, when the rate converges to a positive constant, some providers may still be better off participating in isolation.
To accommodate providers whose rejection rates are lower in isolation, we consider the application of thresholding techniques within the collective. Thresholding allows each provider to set aside a portion of its serving resources to be used exclusively to service its own clients’ requests. We demonstrate that often, by appropriately setting thresholds, all providers in a collective will experience lower rejection rates than when they operate in isolation, even if this property did not hold within the threshold-free version of that collective. Our work demonstrates that, from a performance standpoint, collectives that utilize thresholds often offer a viable, cheaper alternative to overprovisioning or utilizing CDN services.

The rest of the chapter is structured as follows. In Section 2.2, we briefly overview related work. In Section 2.3 we present our general model for server collectives. In Section 2.4 we investigate performance of content delivery services for fixed-rate sessions when considering collective arrangements. We similarly evaluate elastic file transfers in Section 2.5. Section 2.6 evaluates a suite of thresholding techniques. We conclude and elaborate on open issues in Section 2.8.

2.2 Related work

Some works investigate provisioning mechanisms for CDNs with the purpose of a better utilization of resources. The works in [44, 9, 22] study server selection techniques. The studies in [59, 84] investigates the placement of objects (content, as in [59] or application code as in [84]) in the network to reduce delivery latencies. Almeida et al. in [4] evaluate multicast bandwidth consumption for streaming media delivery in CDNs. These works, different from our work, generally assume no competing use of resources.

Several works analyze systems that pool server resources to improve various performance aspects of content delivery. For instance, [34, 112, 111] investigate the practical challenge of maintaining consistency among distributed content replicas. Other studies
[39, 38] investigate load sharing policies. These approaches keep the processing load on a set of hosts relatively balanced while keeping redirection traffic levels low. The Oceano project [54] provides a set of servers that can be spawned and managed to meet additional server resources for customers. After servers are allocated to customers each server is used exclusively by the customers to whom it was allocated and cannot be concurrently shared. An analytical study of systems in which servers are spawned upon cutoff points of a single service demand appears in [49]. A model in which a resource manager distinguishes users into classes that can share a resource first appears in [60]. More recently the study in [29] presented an algorithm to protect such classes from overloads.

The goal in these previous works differs from ours in that there is no notion of individual, competing objectives as there is within a server collective. In other words, in these other works, the only objective is to improve the greater good of the entire system, whereas in our work each provider has its own objective of minimizing the rejection rate of its own content.

The problem of alleviating rapid and unpredictable spikes in request demands (“flash crowds”) has generated attention recently. Jung et al. propose a re-assignment of servers within a CDN infrastructure to handle such events [57]. Wang et al. [113] propose measurement-based server assignment mechanisms within a CDN that distributes loads and proactively can alleviate rapid spikes in demands. Recent proposals in this area [81, 100, 16] solve this problem using peer-to-peer methods, in which clients communicate directly with one another to retrieve the desired content. Here, clients have nothing to gain by serving content. The effectiveness of these approaches simply relies on the goodwill of those who receive content to also transmit the content to others when requested to do so.

The Content Distribution Internetworking (CDI) charter at the IETF has been working towards an initiative whose direction is closer to our work. The CDI working group
concentrates on the definition of requirements and concepts that allow interconnection of CDNs for common content delivery across different content networks [35]. An architecture that implements concepts similar to the CDI initiative is found in [18]. The performance of systems operating under CDI protocols, however, has not yet been analyzed. The analysis of server collectives presented in this chapter can apply to the CDI model.

Very recent works examine aspects that differ from our objective in this study, even though their assumptions are closer to ours. Amini et al. in [7] study heuristic-based algorithms for assignment of content transfers within a set of interconnected CDNs. Balazinska et al. in [14] study load balance mechanisms for federated systems subject to a contract described by sets of upper bound prices defined in bilateral contracts. Ranjan et al. in [85] investigate delay reduction when requests for data centers can be redirected in a wide area network. In particular, the work in [85] models servers as M/G/1/FIFO queueing systems and takes into consideration the replication time.

2.3 Collectives General Model

In this section, we develop the model that allows us to explore the fundamental performance tradeoff that collectives offer to content providers. Namely, we investigate if participating in a collective reduces the rejection rate of requests for a provider’s content. To perform our investigation we develop a model that is simple, elegant, and amenable to a performance analysis.

Our model of a server collective consists of commodities, content providers, clients, servers, and sessions. Commodities are the content/information goods offered by content providers. For instance, multimedia lectures are the commodities of an online course offering. The content provider (or simply provider) is the entity that offers commodities to customers via the Internet. A client requests commodities, and a server interfaces with
the network to deliver commodities to clients. When a server accepts a client request for a commodity, a session (or content transfer) is initiated to deliver the content from the server to the client. A server’s ability to deliver content is constrained by factors such as its processing capabilities (CPU cycle consumption) and its access link bandwidth.\[^{21}\]

To analyze performance of collectives, we assume that a set of providers has already agreed to form a collective and has distributed each commodity to all servers within the collective. A request can be served if there is a server that can immediately process the job associated with that request. Our model assumes that the network core is well-provisioned such that the server’s processing capabilities or its access link to the network are what limit the number of jobs that can be served simultaneously. Hence, a client’s location in the network does not affect the server’s ability to serve that client.

![Diagram of systems in isolation and collective](image)

(a) Systems in isolation. (b) Collective.

Figure 2.1: Collective example compared to systems in isolation.

An example of how a collective, once established, can reduce the rate at which requests for a provider’s content are rejected is depicted in Figure 2.1. Servers \(s_1\), \(s_2\), and \(s_3\) are deployed by three distinct content providers in both Figures 2.1(a) and 2.1(b). The number of sessions a server can host simultaneously is indicated by the number of boxes. Shaded boxes indicate an active session and each clear box is a resource that is available to process a session. In Figure 2.1(a) three different providers operate in isolation (i.e., they do not participate in a collective and do not host one another’s content). The server

\[^{21}\text{In practice, licensing restrictions can place additional limitations on the server side. For instance, RealNetworks [86] offers its basic streaming server (free of charge) with maximum capacity of 5 simultaneous streams. Their $1,999-dollar license server (Helix Server – starter) has maximum capacity of 25 simultaneous streams. The maximum number of sessions is then given by the maximum transmission throughput divided by the average transmission rate of delivery sessions.}\]
labeled $s_3$ cannot service both of the two arriving requests and is forced to drop a request. A logical view of the collective containing these three servers is shown in Figure 2.1(b). Here, server $s_3$ redirects the request it cannot service itself to server $s_2$, which has the capacity to process the job associated with that request. As a result, by participating in the collective, fewer requests for $s_3$'s content are rejected.

<table>
<thead>
<tr>
<th>Table 2.1: Terminology, main variables, and main parameters of the model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{V} = {v_1, v_2, \ldots, v_n}$</td>
</tr>
<tr>
<td>$\mathcal{S} = {s_1, s_2, \ldots, s_n}$</td>
</tr>
<tr>
<td>$\mathcal{V} = {y_1, y_2, \ldots, y_n}$</td>
</tr>
<tr>
<td>$p_{n,i}$</td>
</tr>
<tr>
<td>$N_i$</td>
</tr>
<tr>
<td>$k_i$</td>
</tr>
<tr>
<td>$\lambda_i$</td>
</tr>
<tr>
<td>$B_i$</td>
</tr>
<tr>
<td>$\rho_i$</td>
</tr>
<tr>
<td>$h_i$</td>
</tr>
<tr>
<td>D1–threshold</td>
</tr>
<tr>
<td>D2–threshold</td>
</tr>
<tr>
<td>D3–threshold</td>
</tr>
<tr>
<td>$d_n$</td>
</tr>
</tbody>
</table>

The rejection rate is the metric used to evaluate a provider’s content delivery service. While collectives can be used to handle sudden spikes of demand, our focus is on spikes that last for non-negligible portions of time where the rejection rate can be determined by observing the steady state statistics of the serving system.

We consider a set of content providers $\mathcal{V} = \{y_1, y_2, \ldots, y_n\}$ where a given provider’s commodities are files whose lengths are described by i.i.d. random variables. The set of servers that belong to provider $y_i$ are modeled as a single serving system, $s_i$. We refer to any commodity that originated at provider $y_i$ as $v_i$ without loss of generality. We develop separate models for two classes of content. The first class consists of fixed rate transfers, such as streaming audio or video, where each transfer consumes a fixed amount.
of server bandwidth per unit time, such that the length of a session is independent of the number of files served concurrently by the server. The second class consists of elastic transfers, such as data files, where the amount of bandwidth consumed per transfer per unit time is inversely proportional to the number of files served at that time by the server. In both classes, the number of files that a server will simultaneously transmit is bounded to ensure that transfer rates proceed above a minimum rate. The maximum number \( k \) of simultaneous sessions of a server is the server's capacity. The factors (CPU cycle consumption, access link bandwidth, etc.) that constrain the ability to service sessions typically determine the server's capacity.

We define homogeneity as the property that, for a collective, all providers' intensities are equal and all providers' values of the maximum number of sessions a provider operating in isolation can service (i.e., \( k_i \) for the \( i \)th provider) are equal as well. Thus, a collective is said to be homogeneous when this property holds, otherwise the collective is said heterogeneous. Later in this chapter we will see that this property plays an important role in identifying collectives within which providers achieve smaller rejection rates than are achieved in isolation.

We assume that a provider chooses to participate in a collective as long as the rejection rate of requests for its content is lower than when the provider operates in isolation (comparative criterion).

2.4 Collectives under Fixed-rate Transfers

Here we construct and evaluate a model in which content delivery sessions consist of fixed-rate transfers such as delivery of streaming video.

For fixed rate transfers, we make a simplifying assumption that each commodity (across providers) requires the same rate of transfer, such that each server \( s_i \) is capable of hosting a fixed number, \( k_i \), of sessions simultaneously, where this number is in-
dependent of the set of commodities currently being hosted. The request rate for each provider's commodities, \( v_i \in \mathcal{V} \) is modeled as a Poisson process with rate \( \lambda_i \) and each request receives service immediately if the number of simultaneous sessions is smaller than the maximum \( k_i \). The service times for instances of transfers of commodity \( v_i \) are i.i.d. random variables \( B_i \) with mean \( E[B_i] \), and are independent across the set of all commodities.

Since arrivals of requests for content delivery are modeled as Poisson processes, each serving system is an \( M/G/k/k \) queueing system. If the server in a collective cannot host an arriving request for its commodity, the server forwards the request to an available server (when one exists) in the collective. Otherwise, the request is dropped. Note that if a server operates in isolation, then when it has no additional room to service a request, the request must be dropped. Note that the collective can also be modeled as an \( M/G/k/k \) queueing system with arrival rate \( \sum_{i=1}^{n} \lambda_i \), mean service time \( (\sum_{i=1}^{n} \lambda_i E[B_i]) / (\sum_{i=1}^{n} \lambda_i) \), and that can service up to \( k = \sum_{i=1}^{n} k_i \) sessions simultaneously. The product \( \lambda_i E[B_i] \) is the intensity, \( \rho_i \), of provider \( y_i \).

### 2.4.1 Computing the rejection rate of a collective

We define \( p_{n,i} \) to be the rejection rate of provider \( y_i \)'s content in a collective composed of \( n \) servers (e.g., a set of providers \( \{y_1, y_2, \ldots, y_n\} \) employing servers \( s_1, \ldots, s_n \))\(^2\). A single provider operating in isolation from other providers has rejection rate denoted by \( p_{1,i} \), or simply \( p_1 \). For provider \( y_i \) in isolation, \( i = 1, \ldots, n \), the Erlang loss formula (also known as Erlang B formula) applies directly [89]:

\[
p_1 = \frac{\rho_1^{k_1} / k_1!}{\sum_{j=0}^{k_1} (\rho_1)^j / j!}
\] (2.1)

\(^2\)Whenever \( \forall y_i, y_j \in \mathcal{V}, i \neq j, p_{n,i} = p_{n,j}, p_{n,i} \) can be written simply as \( p_n \).
where $\rho_1 = \lambda_1 E[B_1]$.

We extend this formula to the rejection rate of a two-server collective $p_{2,i}$, $i = 1, 2$. First, we define the random variables $N_i$, $i = 1, 2$, that describe the number of sessions for each of the commodities $v_i$, $i = 1, 2$. Since such a loss system is a symmetric queue [115], the stationary distribution for each state $P(N_1 = x, N_2 = z)$, where $x$ ($z$) is the number of commodities of type $v_1$ ($v_2$) actively being processed, can be expressed in product–form: $P(N_1 = x, N_2 = z) = \pi_x \pi_z c_2$, where $\pi_x = \frac{\rho_1^x}{x!}$, $\pi_z = \frac{\rho_2^z}{z!}$, and $c_2$ is a normalizing constant such that $\sum_{x \geq 0, z \geq 0, x+z \leq k_1+k_2} \pi_x \pi_z c_2 = 1$. Hence we find the rejection rate of a two-server collective is

$$
p_2 = P(N_1 + N_2 = k_1 + k_2) = \sum_{x=0}^{\infty} P(N_1 = k_1 + k_2 - x, N_2 = x)
= \sum_{x=0}^{\infty} \frac{1}{x!} \rho_1^x \frac{1}{(k_1 + k_2 - x)!} \rho_2^{k_1+k_2-x} c_2 = (\rho_1 + \rho_2)^{k_1+k_2} \frac{1}{(k_1 + k_2)!} c_2, \quad (2.2)
$$

where $\rho_i = \lambda_i E[B_i]$, $i = 1, 2$. Noting that (2.2) is in the form of (2.1) with $\rho_1$ replaced by $\rho_1 + \rho_2$ and $k_1$ replaced by $k_1 + k_2$ we can recursively repeat this process to compute the rejection rate for a collective composed of $n$ providers:

$$
p_n = \frac{1}{(\sum_{j=1}^{n} k_j)!} \left( \sum_{j=1}^{n} \rho_j \right) \sum_{j=1}^{n} k_j c_n. \quad (2.3)
$$

where $c_n$ is once more the normalizing constant for this distribution such that

$$
\sum_{x_1 \geq 0, x_2 \geq 0, \ldots, \sum_{j=1}^{n} x_j \leq \sum_{j=1}^{n} k_j} P(N_1 = x_1, \ldots, N_n = x_n) = 1.
$$
2.4.2 Evaluation of Scenarios under Fixed-rate Transfers

We begin by evaluating collectives as a function of number of participating providers and the provider intensities. Figure 2.2(a) plots rejection rates of \( n \)-provider systems \( p_n \), \( n = 2, 5, 10 \), where each server can simultaneously service 100 commodities (\( k_i = 100, i = 1, 2 \)). The results are a direct application of Equation (2.3).

![Figure 2.2(a)](image1)

(a) Rejection rate in collectives scenario compared to servers in isolation.

![Figure 2.2(b)](image2)

(b) Areas of comparison between two-server collectives and servers in isolation.

Figure 2.2: Evaluation of collectives under the model of fixed-rate transfers.

First, we fix the configurations of providers \( y_1, y_2, \ldots, y_{n-1} \), while we vary the traffic intensity of provider \( y_n \). We are interested in observing how the rejection rate of a provider in a collective of size \( n \) is affected when a single provider’s rate is varied. We set \( \rho_i = 65 \) for each provider whose intensity is fixed. This value is chosen such that requests for \( v_i \) exhibit a rejection rate of approximately \( 10^{-5} \) when \( y_i \) operates in isolation, \( 1 \leq i < n \). On the \( x \)-axis, we vary \( \rho_n \), the provider intensity for provider \( y_n \). The \( y \)-axis plots rejection rates for various system configurations. The curve labeled “\( y_n \) in isolation” depicts the rejection rate of requests for commodities \( v_n \) when provider \( y_n \) operates in isolation. The constant line at \( p_1 \approx 10^{-5} \), labeled “\( y_1 \) in isolation”, plots the rejection rate of requests for provider \( y_1 \)’s content when \( y_1 \) operates in isolation (results for providers \( y_2, \ldots, y_{n-1} \) are identical). The remaining curves labeled “\( n \) servers”, \( n = 2, 5, 10 \), depict the rejection rate for all providers that participate in an \( n \)-provider collective. From
the figure, we observe that:

- For the range of values of $\rho_n$ shown, the rejection in the collective for commodity $v_n$ is smaller than its rejection rate in isolation. Therefore, regardless of its own intensity, a provider is willing to participate in a collective when the other providers have sufficiently low intensities.

- Ranges for $\rho_n$ exist, $0 \leq \rho_n \leq 110$, $0 \leq \rho_n \leq 175$, and $0 \leq \rho_n \leq 330$ for $n = 2, 5, 10$, respectively, where the rejection rate in the collective for commodity $v_i$, $i < n$, is smaller than the rejection rate when $s_i$ operates in isolation. Hence, the collective benefits provider $y_i$, $i < n$ even when all other providers in the collective have larger provider intensities. In particular, to achieve a given rejection rate, $\rho_n$ needs to be increased significantly as $n$ increases.

- When $\rho_n$ assumes higher values, the dropping probability for commodities becomes excessive, and according to the described criteria, $y_i$, $1 \leq i \leq n$, would prefer to withdraw its participation from a collective.

Figure 2.2(b) graphically depicts the "areas of participation willingness" for two providers. In this figure we consider a system with two providers, $y_1$ and $y_2$ where $k_i = 100$, $i = 1, 2$. A two-server collective generates a two-dimensional picture as shown in Figure 2.2(b). We vary $\rho_1$ and $\rho_2$ on the $x$- and $y$-axis, respectively. The top curve that goes from the bottom-left to the top-right of the graph is the set of values of $(\rho_1, \rho_2)$ where content provider $y_1$ experiences the same rejection rate regardless of whether it operates in isolation or participates within the collective, i.e., $p_{1,1} = p_2$. Above this curve, $p_{1,1} < p_2$: provider $y_1$'s rejection rate is lower when operating in isolation, and below, $p_{1,1} > p_2$, rejection rate lower when forming a collective with provider $y_2$. Similarly, the lower curve that runs from the bottom left to the top right is formed from the points where $p_{1,2} = p_2$. Below this curve, $p_{1,2} < p_2$, and above, $p_{1,2} > p_2$. 
In Figure 2.2(b), each area is labeled to indicate the “willingness” of \( y_1 \) and \( y_2 \) to form a collective. A label of 'Li', \( i = 1, 2 \), indicates an area in which rejection rate for \( y_i \)’s content is smaller in a collective than in isolation, i.e., \( p_2 < p_{1,i} \). The pair of labels 'L1', 'L2' are replaced by 'L', for simplicity. We see that there is a significant region in which both providers can benefit simultaneously by participating in a collective. In this case, both rejection rates are each lower when both form a collective than when each provider operates in isolation. We refer to a win–only property when for the providers of a collective, every provider’s rejection rate is smaller in the collective than in isolation. We note, however, that both providers tend to benefit only in “near-homogeneous” configurations (in this case defined by the line \( \rho_2 = \rho_1 \)), especially when intensities range from moderate to high. As the difference in intensities widens, the win–only property ceases to hold. In fact, we observe a rapid increase in the rejection rate in Figure 2.2(a) for a collective of 2 servers as \( \rho_n \) is increased in the range \( 85 \leq \rho_n \leq 150 \). In addition, we note that there is no area in which the rejection rates of both content providers are higher in the shared system than in isolation, i.e., at least one provider is willing to participate in the collective in any area marked by an 'X'.

2.4.3 Asymptotic limits of collectives: The \( \rho/k \) factor

We turn to the performance of an \( n \)-server collective as the number of servers \( n \) tends to \( \infty \). In practice a very large collective requires also very large storage units in each of the collective participants since all participants must store all other participants' commodities. There are, however, important insights to be gained from studying the impact on performance as a collective grows. In fact, our results reveal that a rejection rate for providers forming a collective can be close to the asymptotic limit, even for a small number of providers forming the collective.

We assume that there are \( n_c \) different classes of providers, where all provider systems
in the same class exhibit the same provider intensity \( \rho_i \) and have the same bound, \( k_i \), on the number of jobs that can simultaneously be serviced. We let \( f_i \) represent the fraction of providers in class \( i \), \( 1 \leq i \leq n_c \). Equation 2.3 can be reformulated as

\[
p_n = \frac{(\sum_{j=1}^{n_c} (n_j f_j \rho_j)) \sum_{j=1}^{n_c} n_k f_j}{(\sum_{j=1}^{n_c} n_k f_j)!! \sum_{i=0}^{\infty} n_j k_j \left( \sum_{j=1}^{n_c} (n_j f_j \rho_j) \right)^i}
\]

which is simplified via the constants \( \hat{\rho} = \sum_{j=1}^{n_c} f_j \rho_j \) and \( \hat{k} = \sum_{j=1}^{n_c} f_j k_j \) to \( p_n = \frac{(n \hat{\rho})^{n \hat{k}} / (n \hat{k})!}{\sum_{j=0}^{n_c} (n \hat{\rho})^j / j!} \). As \( n \) tends to \( \infty \), the asymptotic limit of the rejection rate is [88]:

\[
\lim_{n \to \infty} p_n = \begin{cases} 
0, & \text{if } \hat{\rho} / \hat{k} \leq 1 \\
1 - \hat{k} / \hat{\rho}, & \text{if } \hat{\rho} / \hat{k} > 1
\end{cases}
\]

(a) Rejection rate when increasing the number of content providers.

(b) Asymptotic rejection rate for \( n \to \infty \).

Figure 2.3: Asymptotic observations on rejection rate for fixed-rate transfer collectives.

Figure 2.3 illustrates the behavior of an \( n \)-server collective as \( n \) grows large. For clarity of presentation, we show only the case where only one class of servers exists, i.e., \( n_c = 1 \) in which each provider's system can simultaneously handle \( k_1 = k = 100 \) transmissions. In this case the average intensity is simply referred as \( \rho = \rho_1 = \hat{\rho} \). In Figure 2.3(a), we vary the number of participating providers along the \( x \)-axis, plotting the rejec-
tion rate along the $y$–axis, where each curve depicts a collective with the corresponding intensity $\rho$. We use Equation 2.3 to plot the curves for $\rho = 95$, $\rho = 100$, and $\rho = 120$. The asymptotic limits are marked with two distinct points. Using Equation 2.5, the asymptotic limit for $\rho = 100$ is effectively 0, and for $\rho = 120$, the limit is approximately $1/6$. The asymptotic limits observed here fit the claims of Equation 2.5. In particular, for $\rho < k$, the rejection rate converges to 0, whereas for $\rho > k$, we observe the rejection rate converging to the limit $1 - k/\rho$. The figure demonstrates (for a homogeneous collection of providers) that the providers benefit from joining a collective with a small number of providers, and that increasing the number of providers in the collective further reduces rejection rates, but at a rate that diminishes quickly.

In Figure 2.3(b) we apply Equation 2.5 to plot the rejection rate of a collective for the limit as the number of providers grows infinitely large as a function of $\rho$. The curve on the left is the asymptotic rejection rate for the case where $k = 20$. We consider only $\rho \geq k = 20$. The curve on the right is for the case where $k = 120$. Again, we consider only $\rho \geq k = 120$. We notice for $k = 120$ a slower increase in the asymptotic rejection rate. We see that the rejection rate converges more slowly and less abruptly (i.e., there is less of a knee) for larger values of $k$.

In conclusion, even when participants in a collective are homogeneous but individually overloaded ($\rho_i > k_i$), their rejection rate for requests for their content remains bounded (below) by $1 - k/\rho$.

2.4.4 Win-only areas and servers’ capacities

In Figure 2.2(b) we observe that both providers $y_1$ and $y_2$ would be better off by forming a collective if their intensities lie within the area given between the upper and lower curves. This area is called a win–only area, as defined previously. In order to observe only variations in intensity we previously consider only $k_1 = k_2 = 100$. In this section
we look at different capacities by varying $k_1$, and $k_2$.

![Graphs showing comparison between collective and server in isolation by varying $k_1 = k_2 = k$.](image1)

(a) Areas of comparison between two-server collectives and servers in isolation by varying $k_1 = k_2 = k$.

![Graphs showing comparison between two-server collectives and servers in isolation by varying $k_1 = k_2 = k$ and normalizing the intensity.](image2)

(b) Areas of comparison between two-server collectives and servers in isolation by varying $k_1 = k_2 = k$ and normalizing the intensity.

Figure 2.4: Evaluation of collectives under the model of fixed-rate transfers by varying values of capacities $k_1$ and $k_2$.

In Figure 2.4(a) we show the curves where $k_1 = k_2 = k$, when $k = 10$, $k = 20$, $k = 50$, and $k = 100$. The win-only areas for the cases where $k = 10$, $k = 20$, $k = 50$, and $k = 100$ are observed between the upper and lower curves labeled “$k = 10$”, “$k = 20$”, “$k = 50$”, and “$k = 100$”, respectively. In Figure 2.4(b) we apply the same set of results observed in Figure 2.4(a) but now the intensities are shown by normalized values defined by $\rho_1/k_1$ and $\rho_2/k_2$. This allows us to compare all cases from $k = 10$ to $k = 100$ when the intensities are at equal proportions to the capacities. For example, the intensity equals the capacity at the server when $\rho_1/k_1 = 1$. We observe that the “bulb” that forms the win-only areas appears similar from $k = 10$ to $k = 100$. The gap between the curves for higher values of $\rho_1/k_1$ and $\rho_2/k_2$ is larger for smaller values of $k$, such as $k = 10$, than for larger values, such as $k = 100$. This happens because the configuration in isolation results in higher rejection rates for equal values of $\rho_1/k_1$ and equal values of $\rho_2/k_2$, as $k$ decreases.

In Figures 2.5(a) and 2.5(b) we observe the results from the configurations in which provider $y_1$ deploys a server with $k_1 = 20$ and $k_1 = 50$, respectively, and $y_2$ deploys a
Figure 2.5: Distinct configurations by varying $k_1$.

server with $k_2 = 100$ (in both figures). We observe that the win-only areas tilt to the side of the axis relative to the server that has more capacity. This is expected since for lower intensities the rejection rate for provider $y_1$ increases as $k_1$ decreases, so the "win" area for provider $y_1$ moves up. For provider $y_2$, the "win" area is delimited by the lower curve, which tilts to the left as $k_1$ decreases. This is a natural consequence of the fact that the capacity in the collective is smaller for smaller $k_1$.

In Figure 2.6 we present the results at normalized values for $k_2 = 100$ and $k_1$ in the set $\{10, 20, 50\}$. The lower curves are almost unchanged as $k_1$ decreases. The upper curves, however, move up as $k_1$ decreases, which causes the "bulbs" that form the win-only areas to slightly expand. This happens because of the same reason observed for the expansion of the areas in Figure 2.4(b), essentially, because the rejection rates in isolation for provider $y_1$ increase as $k_1$ decreases. Here again deviations from the line given by $\rho_1/k_1 = \rho_2/k_2$ makes one of the providers better off operating in isolation.

### 2.5 Collectives for Elastic Transfers

In the previous section we focused on performance benefits for providers forming a collective where their content distribution services are fixed-rate transfers. In this section,
we focus on performance benefits for providers forming a collective where their content distribution services are elastic transfers such as file transfers using TCP/IP. We consider collectives that receive traffic whose service rate is proportional to $1/w_n$, where $w_n$ is the number of customers served simultaneously across all $n$ servers that comprise the collective. For such a model, load is equally balanced across the collective. This can be accomplished in practice, for instance, with parallel downloading technology [20], which is commonplace today in peer-to-peer downloading tools such as Kazaa.

We again consider a model in which client arrivals to each provider are described by a Poisson process and the load imposed by each commodity is described by a general distribution. Using the same argument as before, we collapse the model to the case where each provider offers a single commodity and uses a single server. The processor sharing (PS) service model applies here since the transmission rate is inversely proportional to the number of requests in service. We assume that each server $s_i$ bounds the minimum rate at which it will transmit data to clients by bounding the number of clients accepted by a constant, $k_i$, and will turn away (or redirect) any additional clients requesting service. In the context of parallel downloading, the sum of all fractional components serviced should add up to $\sum_{i=1}^{n} k_i$. Therefore, a server is modeled as an $M/G/1/k_i/PS$ queue and a collective modeled by an $M/G/1/k/PS$ queue, where $k = \sum_{i=1}^{n} k_i$. We assume,
without loss of generality, that work for delivering \( y_t \)'s content in isolation is processed at a rate equal to the maximum number of simultaneous sessions \( k_t \), and for a collective at rate \( k \). As before, we assume that jobs are never queued when service is unavailable, but are simply turned away (dropped). In addition to rejection rate as a metric, we also measure job completion time.

The amount of work necessary for a job processed by the server collective is a random variable \( U \). For an \( n \)-server collective, the servicing time \( \hat{B}_n \) for each individual job in the collective is \( \hat{B}_n = U/k \). Since the servicing time \( \hat{B}_1 \) obtained with a single provider in isolation is \( \hat{B}_1 = U/k_1 \), \( \hat{B}_n \) is a function of the servicing time, \( \hat{B}_1 \), such that \( \hat{B}_n = k_1 \hat{B}_1/k \). While the distribution of an unbounded queueing system that uses a processor sharing discipline is known to be geometric [89], for a finite queueing system we find the following law:

\[
p_n = (\lambda E[\hat{B}_n])^k \frac{1 - \lambda E[\hat{B}_n]}{1 - (\lambda E[\hat{B}_n])^{k+1}}
\]

(2.6)

where \( \lambda = \sum_{j=1}^{n} \lambda_j \) (\( \lambda_j \) is the request rate for commodity \( v_j \)), and \( E[\hat{B}_n] \) is the mean servicing time of a job processed by the collective. Here, the intensity is \( \rho = \lambda E[U] \), where \( E[U] = kE[\hat{B}_n] \).

Since for elastic transfers the service rate is inversely proportional to the number of simultaneous transfers, the completion time is also a metric of interest. For a collective formed from \( n \) providers, the expected completion time of a session \( d_n \) is obtained using Little's Law, in terms of the fraction of requests accepted to the system \((1 - p_n)\lambda \). We let the number of concurrent, servicing sessions be a random variable \( N \). Using Little’s law we find

\[
d_n = \frac{\bar{w}_n}{(1 - p_n)\lambda},
\]

(2.7)

where \( \bar{w}_n \) is the expected number of simultaneous sessions, \( \bar{w}_n = \sum_{z=0}^{k} zP(N = z) \).
2.5.1 Numerical evaluation

(a) Rejection rate under a processor sharing discipline.

(b) Mean completion time under a processor sharing discipline.

Figure 2.7: Evaluation of a collective for elastic transfers.

Here we study conditions under which providers are willing participants within a collective that consist of providers whose transfer rates are elastic. As in Section 2.3 we again compare a provider’s rejection rate when forming a collective with other providers to its rejection rate when operating in isolation. For instance, Figure 2.7 illustrates a scenario in which we vary \( \rho_1 \) and maintain \( \rho_2 \) fixed at \( \rho_2 = 91 \). Here, \( k_1 = k_2 = 100 \).

Figure 2.7(a) and 2.7(b) plot the rejection rate and average completion time, respectively, of the providers in collectives as well as of providers in isolation as a function of \( \rho_1 \). The curves labeled “collective”, “\( y_1 \) in isolation”, and “\( y_2 \) in isolation” respectively plot \( p_2 \), \( p_{1,1} \), and \( p_{1,2} \) (the last being constant). We observe the rejection rate of providers within the collective to be almost 4 orders of magnitude smaller than \( p_{1,1} \) when \( \rho_1 \) and \( \rho_2 \) are approximately equal. However, the benefits become marginal with increasing \( |\rho_1 - \rho_2| \).

This again supports the intuition that a collective is useful for elastic transfers only when the intensities imposed by providers are approximately the same.

We observe from Figure 2.7(b) that the conclusions for completion time similar to those obtained for the rejection rate. We vary the intensity \( \rho_1 \) along the \( x \)-axis. The average completion time is shown along the \( y \)-axis. The curves labeled “collective”, “\( y_1 \)
in isolation”, and “y₂ in isolation” respectively plot the average completion time for each provider within the collective, for provider y₁ in isolation, and provider y₂ in isolation. The average completion time for a request to provider y₂’s commodity is reduced significantly when \( \rho_1 \leq 80 \). This reduction occurs because jobs for provider y₂’s content are likely to receive treatment from y₁’s underutilized resources. In the range \( 80 \leq \rho_1 < 100 \), provider y₁’s average completion time is reduced significantly. Thus, this range is important because provider y₁ and y₂ both benefit from participating in the collective. For \( \rho_1 > 100 \), provider y₁ can still obtain dramatically lower completion times within a collective, as low as a sixth of its completion time in isolation, but provider y₂’s average completion time is greater than in isolation. Therefore provider y₂ will not be willing to form a collective.

![Figure 2.8: Areas of benefit for collectives under processor sharing discipline](image)

Figure 2.8 depicts willingness areas for elastic transfer services computed via application of Equation 2.6. The parameters \( \rho_1 \) and \( \rho_2 \) are varied respectively along the x– and y–axis. We use the same labeling convention as in Section 2.4.2. In comparison to Figure 2.2(b), we see that a collective in elastic environments favors both providers for a wider variation of their respective intensities when both intensities are small (i.e., the bubble in the bottom-left corner is bigger). However, when intensities are large, the difference in provider intensities over which both providers are willing to form a collective is reduced. Intuitively, this may be due to the fact that when the system is under high loads,
the average completion time of a job increases. Bringing in additional capacity (but with a proportional load) does not reduce completion times of admitted jobs significantly when load is high. It will, however, reduce completion times when load is light. Since a collective formation is most useful when intensities are high (e.g., in Figure 2.8, the bulb does not stretch as much as in Figure 2.2), we conclude that collectives can tolerate more heterogeneity in systems when servicing fixed-rate requests than when servicing elastic-rate requests.

2.6 Resource Bounding with Thresholds

Here, we evaluate thresholding techniques as a means to limit the amount of server resources a provider contributes to a collective. We show how thresholding can be used to bound rejection rates of providers, thereby encouraging participation within a collective system. Such schemes can be used in either fixed-rate or elastic transfers. Here, we perform the analysis using fixed-rate transfers.

Let $h_i$ be a threshold, $0 \leq h_i \leq k_i$ for server $s_i$ such that $s_i$ refuses any requests to service other provider's commodities whenever it is actively servicing $h_i$ commodities of other providers. This guarantees that the provider will maintain space to simultaneously service at least $k_i - h_i$ requests of its own commodity at any given time. We call this threshold type D1. We also evaluate a second thresholding technique, named D2-thresholding. A D2 type threshold denies a request at server $s_i$ for another provider's commodity $v_j$, $j \neq i$, whenever the available space for commodity transmission at server $s_i$ falls below $h_i$ (i.e., $k_i - x_i$ where $x_i$ is the number of sessions in service). An advantage of D2 over D1–thresholding is a stronger protection for a provider service when its intensity is high, since it can reject requests for commodities other than its own even if no job associated with these commodities is in service. For both types of thresholding, setting $h_i = 0, 1 \leq i \leq n$, is equivalent to $n$ providers operating in isolation and setting $h_i = k_i$
for all $i$ is equivalent to a collective described in previous sections where providers fully share their resources.

We also consider a third thresholding technique, D3-thresholding, in which a provider’s threshold is a function of its average number $a$ of sessions redirected to other provider’s servers. Under D3-thresholding, a provider accepts a number $h$ of sessions for content other than its own, such that $h \leq a + \bar{a}$, where $\bar{a}$ is a tolerance value beyond the average number $a$ of redirected sessions. This type of tolerance value allows providers to redirect requests when their average number of redirected sessions is greater than $a$ but still within a range defined by the tolerance value $\bar{a}$. Besides allowing for variation in the level of tolerance set by the value of $\bar{a}$, this formulation permits a “bootstrap” of the process of redirecting sessions when the average number of redirected sessions is zero, i.e., when $a = 0$. If a provider’s ability to redirect requests to other providers’ servers is bounded by $a$, then when $a$ equals zero a provider does not permit utilization of its resources.

All three types of thresholding can be implemented with or without switching. When switching is implemented, a job for provider $i$’s that is assigned to a server $j \neq i$ can be moved back to $i$ as soon as there is an available process at server $i$, i.e., jobs are always moved to their preferred server when there is room. In practice enabling switching entails additional overhead, but for analytical purposes the model is more tractable. We shall see shortly (comparing simulation results) that enabling switching has little impact on rejection rate, so that the analytical results obtained when switching is allowed are a good approximation for models in which switching is not permitted.

2.6.1 Analytical evaluation of D1 thresholding

D1-thresholding is a coordinate-convex sharing policy, thus a product-form solution is still valid under general service time distributions [29]. However, for a simplified, comprehensible analysis, we assume that service times are exponentially distributed at rate
\( \mu_i, 1 \leq i \leq n \). This allows us to model the 2-provider collective as a truncated Markov chain with states described by the pair \((N_1, N_2)\), where \(N_i\) is the number of sessions servicing commodity \(u_i\) in the system, \(i = 1, 2, 0 \leq N_1 \leq k_1 + \min(k_2 - N_2, h_2)\), and \(N_2 \leq k_2 + \min(k_1 - N_1, h_1)\). A crucial difference from previous models is that here, since a provider's decision to accept a request depends on whether the commodity being requested belongs to the provider, the rejection rates for the differing commodities can differ within a collective. The Markov chain transitions are as follows:

- From \((x - 1, z)\) to \((x, z)\) with rate \(\lambda_1\), for \(1 \leq x \leq k_1 + \min(k_2 - z, h_2)\).
- From \((x, z)\) to \((x - 1, z)\) with rate \(x \mu_1\), for \(1 \leq x \leq k_1 + \min(k_2 - z, h_2)\).
- From \((x, z - 1)\) to \((x, z)\) with rate \(\lambda_2\), for \(1 \leq z \leq k_2 + \min(k_1 - x, h_1)\).
- From \((x, z)\) to \((x, z - 1)\) with rate \(z \mu_2\), for \(1 \leq z \leq k_2 + \min(k_1 - x, h_1)\).

A product-form solution is derived: \(P(N_1 = x, N_2 = z) = \pi_{x,z} = \pi_x \pi_z c_2\), where \(\pi_x = \rho_1^x / x!\), \(\pi_z = \rho_2^z / z!\), and \(c_2\) is a normalizing constant such that

\[
\sum_{x=0}^{k_1+h_2} \sum_{z=0}^{k_2+\min(h_1,k_1-x)} \pi_{x,z} = 1.
\]

We use the probabilities \(\pi_{x,z}\) to compute the probabilities of all states for which \(N_i = k_i + \min(k_j - w, h_j)\), given that \(N_j = w, i = 1, 2, j = 1, 2, j \neq i\). The rejection rate of provider \(y_i\)'s content in the collective, \(p_{2,1}\), is:

\[
p_{2,1} = \sum_{x=k_i-h_i}^{k_i+h_j} \pi_{x,k_i+k_j-x} + \sum_{z=0}^{k_j-h_j-1} \pi_{k_i+h_j,z}
\]

\[
= \sum_{x=k_i-h_i}^{k_i+h_j} \frac{(\lambda_i/\mu_i)^x (\lambda_j/\mu_j)^{k_i+k_j-x} c_2}{x! (k_i+k_j-x)!} + \frac{(\lambda_i/\mu_i)^{k_i+h_j} (\lambda_j/\mu_j)^{k_j-h_j-1} c_2}{(k_i+h_j)! z!}.
\]  

\((2.8)\)

\(^3\)Note that our reduction of a provider's server system to a single server with a single commodity still holds without loss of generality.
The rejection rate of provider $y_2$'s content in the collective, $p_{2,2}$, is obtained in analogous manner.

![Figure 2.9: Respective rejection rate experienced on requests for providers $y_1$'s and $y_2$'s content and different thresholds.](image)

Figure 2.9 depicts respective rejection rates experienced for requests of content of both providers $y_1$ and $y_2$ for various intensities and threshold levels under D1–thresholding with switching, where providers $y_1$ and $y_2$ apply the same threshold, i.e., $h_1 = h_2$. Each of them has a server with total capacity for $k_i = 100$, $i \in \{1, 2\}$, concurrent sessions. On the $x$–axis, we vary $\rho_1$. Instead of fixing $\rho_2$, we set $\rho_2 = 0.2\rho_1$ such that the intensities that both providers contribute to the collective increase along the $x$–axis, but $y_1$’s intensity remains much larger than that of $y_2$. The various curves depict rejection rates for the two commodities for differing threshold levels. The curves for the systems in isolation are represented with thicker lines. The curve labeled “totally shared” is the rejection rate for both commodity requests when the thresholds are set to maximum values $h_1 = h_2 = 100$. The remaining curves’ labels indicate the provider whose content’s rejection rate is plotted and the value to which the threshold is set.
The most important conclusion here is that changing the threshold value can lead to significant variations in rejection rate for a collective. In fact, when the intensity of a provider is small enough that the provider can meet its desired rejection rate operating by itself, that provider can then use thresholds in a collective to allow other content providers the use of its resources without raising its own rejection rate above the undesired level.

2.6.2 Comparison of Thresholding Techniques

We resort to simulation to evaluate non-switching and D2 and D3–threshold configurations. We model arrivals by a Poisson process, but service times here are described by a lognormal distribution, as has been observed in practice [5, 80, 28, 55]. The probability density function of the lognormal distribution is given by \( \frac{e^{-(\log(x) - \mu)^2/(2\sigma^2)}}{x\sigma \sqrt{2\pi}} \), where \( \log(x) \) is the natural logarithm and \( \mu \) and \( \sigma \) are the standard parameters used within the lognormal distribution. We used the mean and standard deviation of 26 and 46 (minutes) observed in [5], respectively, to derive the parameters \( \mu \) and \( \sigma \).

![Graphs showing rejection rate vs \( \rho_1 \) and \( \rho_2 \)]

(a) Rejection rate of provider \( y_1 \)'s content. (b) Rejection rate of provider \( y_2 \)'s content.

Figure 2.10: The use of D1–thresholding and D2–thresholding/switching.

Based on measurements from [5], we conduct simulations with \( \lambda = 3.5 \) requests per minute and \( E[B] = 26 \) minutes giving a value \( \rho_2 = 91 \) for provider \( y_2 \)'s content. Fig-
Figure 2.10 plots rejection rates obtained from the previous analysis (Equation 2.8, for the case of D1 thresholding with switching) and simulations (for the other cases) in which \( h_i = 40 \) for \( i = 1, 2 \). Figure 2.10(a) plots rejection rates for provider \( y_1 \)'s content, and Figure 2.10(b) plots rejection rates for provider \( y_2 \)'s content. Here \( \rho_1 \) is varied along the \( x \)-axis in both Figure 2.10(a) and Figure 2.10(b). The curves in each figure labeled “D1+switching”, “D1+non-switching”, “D2+switching”, and “D2+non-switching” depict the various collective thresholding configurations formed by alternating between the use of non-switching and switching methods, and between the use of D1 and D2 thresholding techniques.

In Figure 2.10(a), we observe little difference in rejection rate for provider \( y_1 \)'s content for the varied configurations. In Figure 2.10(b), we observe the rejection rates of provider \( y_2 \)'s content. The horizontal line depicts the rejection rate for provider \( y_2 \)'s content when \( y_2 \) operates in isolation. For \( \rho_1 < 40 \) the two curves with sharp increases correspond to the collective with D2–thresholds. The two curves are indistinguishable except for \( \rho_1 > 100 \) where the switching system exhibits a rejection rate that is slightly lower than the non-switching system. The two curves with sharp increases in the range \( 60 < \rho_1 < 100 \) correspond to the collective with D1–thresholds. When operating in the collective, the rejection rate for provider \( y_2 \)'s content drops by as much as three orders of magnitude in the range where \( \rho_1 < 50 \) using D2 and \( \rho_1 < 100 \) using D1–thresholds. In the range \( \rho_1 > 100 \) the rejection rates for provider \( y_2 \) converges to the rejection rate when provider \( y_2 \) operates in isolation. This is because \( \rho_2 \) is sufficiently high to make it unlikely that the number of sessions delivering provider \( y_2 \)'s content places the available space at \( y_2 \)'s
server below its threshold $h_2$. In contrast, when using D1-thresholding the exhibited rejection rates are much lower than those exhibited when using D2 thresholding when $\rho_1 < 100$.

We further observe that there is little difference in the results obtained when ongoing session switching is enabled from when it is not. This suggests that analytical results for rejection rate from a switching system can be used to approximate the rejection rate within non-switching systems.

![Graphs](a) Rejection rate of provider $y_1$'s content. (b) Rejection rate of provider $y_2$'s content.

**Figure 2.11:** Comparison of D1–thresholding and D3–thresholding/switching.

Figure 2.11 depicts a comparison between the performance of D1 and D3–thresholding. We apply the same simulation experiments used to compare performance of D1 and D2 thresholding. Provider $y_2$’s intensity is kept constant at $\rho_2 = 91$, while $\rho_1$ is varied. To remain consistent with previous parameters, both providers $y_1$ and $y_2$ are set to configurations in which $k_i = 100$ and $h_i = 40$, $i \in \{1, 2\}$. The tolerance value is $\tilde{a} = 10$. Figure 2.11(a) shows the rejection rate of provider $y_1$’s content, whereas Figure 2.11(b) shows the rejection rate of provider $y_2$’s content.

We observe that the main difference in the performance obtained with D3–thresholding
compared to using D1–thresholding (and also to using D2–thresholding) is that D3–thresholding only allows a reduction of rejection rate of provider $y_2$’s content by no more than one order of magnitude in the range where $\rho_1$ is low, such as $\rho_1 < 50$. In this range provider $y_2$ can only reduce its rejection rate up to a limit approximately given by $\bar{a}$. This happens because for low values of $\rho_1$, provider $y_2$ does not redirect much of its incoming requests, hence the average number $a$ of redirected sessions for provider $y_1$ is low and the threshold is limited by $a + \bar{a}$. In contrast, in the range $50 < \rho_1 < 80$ provider $y_1$ can offer larger thresholds to provider $y_2$. As a result, we observe larger reduction of the rejection rate of provider $y_2$’s content from its rejection rate when provider $y_2$ is in isolation. For high values of $\rho_1$ rejection rates obtained with D3–thresholding for both providers are similar to ones respectively obtained using D1–thresholding. This comes as a result of the use of the extra allowance $\bar{a}$ which reserves space in providers for content different than their own commodities in the same fashion as D1–thresholding does.

2.6.3 Extending Heterogeneity

Thresholding encourages providers to participate in collectives with performance benefits where the same providers are unwilling to participate in a collective without any thresholding. Applying the comparison between rejection rate obtained in isolation and the rejection rate obtained in a collective, we can find the potential areas of interest for two providers $y_1$, and $y_2$ when using the same pair of thresholds. We wish to determine if provider $y_1$ performs better in isolation than in a collective with $y_2$, and vice-versa, given their respective intensities $\rho_1$ and $\rho_2$ and threshold values $h_1$ and $h_2$ (D1–thresholding).
Figure 2.12: Extending heterogeneity using thresholds.

We perform an exploration similar in form to that used in Section 2.4.2, but now the collectives are formed with restrictions given by the two thresholds, $h_1$ and $h_2$, and the rejection rates are obtained using Equation 2.8. We show in Figure 2.12 how forming a collective can bring benefits to both providers under more heterogeneous conditions for providers $y_1$ and $y_2$. The intensity $\rho_1$ is varied along the x-axis and the intensity $\rho_2$ is plotted along the y-axis. The area between the curves labeled '$h_1 = h_2 = 100$' is the set of values for $\rho_1$ and $\rho_2$ where both $y_1$ and $y_2$ share without restrictions. The area between curves labeled '$h_1 = h_2 = 4$', and '$h_1 = h_2 = 2$' indicate values of $\rho_1$ and $\rho_2$ for which both providers are using the same value of threshold, i.e., only up to 4 slots or 2 slots, respectively, can be used to serve a content for the other provider. All these areas have in common that both providers have smaller rejection rates than their rejection rates in isolation, i.e. these are "win-only" areas. We conclude from the shown curves that thresholding indeed extends the heterogeneity tolerated for establishing collectives. However, the two providers do not necessarily need to have the same threshold values.
2.6.4 Optimal Thresholding

Motivated by the results in previous sections, we consider an ideal scenario in which a provider, \( y_i \), selects its optimal threshold as a function of the provider intensities imposed on the collective. By doing so it contributes the maximum amount of its own server resources (i.e., the highest threshold possible) without the rejection rate for its own commodity, \( v_i \), exceeding a value \( l_i \).

We apply our analysis of D1-type thresholding systems with switching to a 2-provider collective in which providers \( y_1 \) and \( y_2 \) accept a maximum of \( k_1 = 100 \) and \( k_2 = 20 \) concurrent sessions, respectively. Provider \( y_1 \) receives a fixed intensity of \( \rho_1 = 20 \). Provider \( y_1 \) adjusts its threshold \( h_1 \) to the maximum integer value (via Equation 2.8) such that the rejection rate for provider \( y_1 \)'s content remains below the value of \( l_1 = 10^{-5} \). In this case, provider \( y_1 \) relaxes its condition for willingness to participate in the collective, and requires only that its content’s rejection rate remains below \( l_1 \). Provider \( y_2 \) enables its server to fully share its resources, i.e., \( h_2 = k_2 = 20 \).

![Figure 2.13: Optimal thresholding.](image1)

(a) Threshold adjustment.

(b) Rejection rates.

In Figure 2.13(a) we vary \( \rho_2 \) in the x-axis and plot the largest value of \( h_1 \) along the
$y$-axis as a function that maintains $p_{2,1} < l_1$. We see that for $\rho_2 \leq 60$, the threshold remains at 100. As $\rho_2$ crosses 60, the threshold drops rapidly, then continues to reduce, but at a much slower rate. In Figure 2.13(b) we vary $\rho_2$ along the $x$–axis, and the rejection rate along the $y$–axis. Figure 2.13(b) shows rejection rates of various configurations as a function of $\rho_2$. The left-most curve plots the rejection rate of provider $y_2$'s content when $y_2$ operates in isolation (obtained from Equation 2.1). The remaining three curves (which differ only when $\rho_2 > 60$) plot, from top to bottom, the rejection rate of provider $y_2$'s content when participating in the collective with optimal thresholding, the rejection rate for all commodities when participating in the collective without thresholding (obtained from Equation 2.2), and the rejection rate of provider $y_1$'s content when participating in the collective with the optimal thresholding.

The bottom curve verifies that with thresholding, rejection rates of provider $y_1$'s content remain below $l_1 = 10^{-5}$. By comparing the remaining two curves from the collective to the curve for the case where $y_2$ is in isolation, we see that, even with thresholding, participating in the collective significantly reduces provider $y_2$'s rejection rate. We see that, while thresholding increases the rejection rate for provider's $y_2$'s content in comparison to a threshold-free collective, provider $y_1$ is willing to participate in the collective only when the thresholding is applied, and provider $y_2$ experiences a rejection rate that is orders of magnitude smaller than if $y_2$ operates in isolation. Such adjustments permit a provider to set a target rejection rate to not be exceeded.
2.7 Experimentation of Collectives for a Video Delivery Service: proof of concept

We proceed to describe the SERES streaming media architecture, a platform for instantiating server collectives comprised of streaming media servers. We describe the protocol used to let a provider join and share its content within a collective. To protect each provider's interests, SERES incorporates a D1-thresholding mechanism, which allows a provider to isolate a portion of its serving resources and reserve them exclusively for servicing requests for its own content. Doing so helps assure that, even when other providers place heavy loads on the SERES system, a provider still has enough of its own resources available to service the expected demands of its own clients. We present experimental results of a prototype of SERES upon an actual distributed testbed that shows that SERES does reduce rejection rates of client requests for each participating provider.

SERES implements additional signaling capabilities to support traditional client-server architectures of streaming media services, e.g. enabled via RTSP protocol. Servers forming a collective are called SERES participants. A simple redirection mechanism permits a provider to provision its resource utilization at expected demand or to configure limits for its resource utilization. For instance, a bound can be placed on the aggregate bandwidth that is used to serve client requests. We refer to this bound as the Maximum Local Resource Usage (MLRU).
2.7.1 Thresholding

When a server redirects many requests in a relatively short period of time, a server that receives an excessive number of redirected requests might perceive its own content delivery service affected. The content that a provider’s server hosts originally for its profit is referred to as primary content, whereas any other content hosted in behalf of other SERES participants is non-primary content. A provider thus configures thresholds that limit the number of non-primary related sessions. The ability of configuration of thresholds of non-primary content while still contributing to the collective in SERES is referred as thresholding.

Therefore, for new non-primary requests an admission rule is defined as follows. Two possible outcomes exist: either accept or reject. Let the variable stats and nonp.stats account for current state information of the supported service about primary content and con-primary content, respectively. A variable m1ru stands for the maximum local resource usage. A variable threshold contains the current threshold limits imposed to requests of non-primary content. As a general rule, for requests of non-primary content an accept outcome is a result of a true boolean statement (stats < m1ru) AND (nonp.stats < threshold), and a reject outcome results, otherwise. The values of stats and nonp.stats are typically the aggregate of utilized bandwidth, and/or number of sessions, and/or CPU consumption. In case of a single metric being observed, the values of stats, m1ru, nonp.stats, and threshold are represented by scalar values. When more than one metric is of interest, each of these variables can be represented by a vector \( (x_1, x_2, \ldots x_n) \) comprising the interested values.
In this last case the "<" operation needs to be extended to the following: \( a < b \) results
TRUE, if \( (a_1 < b_1) \) AND \( (a_2 < b_2) \) AND ... \( (a_n < b_n) \) and results FALSE, otherwise.

2.7.2 Implementation and Experiments

We implemented the SERES architecture as explained in Section 2.7 to support streaming
media services hosted by the Darwin Streaming Server[10], which is capable to deliver
streaming video in the Quicktime codecs and also MPEG-4. Supported transport protocols include UDP/RTP, TCP and HTTP. Darwin uses also the RTSP protocol (RFC 2236
[92]) for its streaming media signaling layer. Our implementation is developed in C++
language and has been tested in Linux/Red-Hat and MS-Windows XP/2000. SERES
messages are exchanged over TCP connections, using a port number common to all participants of the collective.

We use as performance data of the server a set of measurements computed periodically by the Darwin server: current number of RTSP sessions (NumberSessions),
corresponding to a current number of simultaneous RTSP sessions in service; current bandwidth (CurrentBandwidth), corresponding to the number of RTP packet bytes
transmitted during the sampled time interval divided by its length; and percentage of CPU
consumption (CPUPercent), the relative amount of the processing power is used by the
Server process in proportion to the total CPU capacity. A provider can typically configure
both the server's MLRU values and the threshold values specified by a vector comparable
to the sampled performance data: (CurrentBandwidth, NumberConnections,
and CPUPercent).
Additional material describing the implementation is found in the Appendix B.

2.7.3 Methodology

We conduct experiments with servers either participating in a SERES collective or remaining in isolation. The experiments are performed using synthetic streaming media workloads, because we do not have traces of actual streaming media workloads. Traces of representative actual services are indeed not generally freely available. We resort to GISMO[55], a workload generator of streaming media traffic developed at Boston University. GISMO generates streaming-media workloads using a model constructed from empirical description of such systems. We run a scheduler of instances of RTSP clients that uses a GISMO-generated workload trace as its request base. We developed the scheduler, which uses openRTSP [69] to instantiate RTSP clients. open RTSP is a simple RTSP client capable of handling RTP/RTCP streams.

To form our testbed, we run SERES server instances located at the corresponding IP addresses of planetlab3.comet.columbia.edu (columbia, for short), located at our laboratory, planetlab3.xeno.cl.cam.ac.uk (cambridge), physically located at University of Cambridge, UK, and planetlab1.it.uts.edu.au (uts), located at University of Technology Sydney, Australia.\footnote{These hosts run in conjunction of the Planet Lab initiative [82].} The clients are located at three local machines in our lab domain (alcyone, atlas, and planetlab1, all in domain comet.columbia.edu). Both servers and clients run Linux as operating system. For any synthetic workload, all the clients are instances at a same machine. This approach is
chosen for simplicity and also to avoid the situation of having more than one client as traffic sources, exhibiting different round trip times to the server. Otherwise, he effect would be a workload in the server different from the one actually generated. The average length time of a session for the workload traces of columbia, cambridge and uts servers are respectively 36 minutes, 32 minutes, and 36 minutes. The servers are instrumented to count the number of events such as accepted requests, rejected requests, number of requests for redirections, etc., onto a log file to be analyzed postmortem. The servers are also instrumented to measure and update local performance data at regularly spaced intervals of 1000 ms. Each of the servers services as primary content one streaming media presentation, a Quicktime file encoded at transmission rate of 300 kbit/s and with total duration 70 minutes. Each of these presentations consist of two streams, a video stream and an audio stream.

In our first experiment lasting six hours approximately the three servers form a SERES collective. Using the same traces we re-enact the workloads to conduct three experiments (also lasts for six hours each) with each server now running separately from the others, i.e., in isolation. Each server sets the MLRU at (10 Mbit/s, 80 sessions, 70% of CPU) and its threshold at (1 Mbit/s, 20 sessions, 20% of CPU). SERES collective creation occurs when a first server starts running with no collective attachment and subsequent servers call the first server or one of the then-joined SERES servers to join the collective.
Table 2.2: Number of occurrences of events and rejection rate during experiments

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Events</th>
<th>Columbia</th>
<th>Cambridge</th>
<th>UTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERES</td>
<td>accepted requests</td>
<td>344</td>
<td>345</td>
<td>365</td>
</tr>
<tr>
<td></td>
<td>redirections</td>
<td>22</td>
<td>28</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>rejected requests</td>
<td>19</td>
<td>24</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>rejection rate</td>
<td>5.2%</td>
<td>6.7%</td>
<td>5.1%</td>
</tr>
<tr>
<td>isolation</td>
<td>accepted requests</td>
<td>337</td>
<td>315</td>
<td>330</td>
</tr>
<tr>
<td></td>
<td>rejected requests</td>
<td>27</td>
<td>46</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>rejection rate</td>
<td>7.4%</td>
<td>12.7%</td>
<td>8.8%</td>
</tr>
</tbody>
</table>

Figure 2.14: Bandwidth consumed by servers under SERES.
2.7.4 Results

Table 2.2 contains the number of occurrences of the following events for the three servers in both the experiments in isolation and the one with the three of them forming a collective: primary RTSP requests accepted at one of the SERES participants as event "accepted requests"; successful redirection requests as event "redirections" (it does not apply for the experiment in isolation though); events of rejected calls for primary content as "rejected requests". The third column shows the number of times each event is registered for the columbia server. This server had 344 primary requests accepted, 19 primary requests rejected, 22 successful youfree inquiries, and 27 rejected requests when running in isolation. The rejection rate when using SERES was 5.2% while the rejection rate of calls for the columbia's content when in isolation was 7.4%. The fourth column indicates the number of occurrences of events for the cambridge server. The cambridge server experienced 345 accepted primary requests, 24 rejected requests for non-primary content, 28 successful YOUFREE inquiries, and 46 rejected requests when experimenting in isolation. For the cambridge server, a comparison between rejection rates in isolation and joined to the collective shows a reduction from 12.7% to 6.7%. The fifth column shows the ucs server data, which registered 365 accepted requests for primary content, 19 rejected requests in SERES, 22 successful youfree inquiries, and 32 rejected requests in isolation. For the ucs's content the rejection rate reduction came from 8.8% in isolation to 5.1% in the experiment of the collective. Therefore, all three servers realize reduction of rejection rates when compared the performance obtained when participating of a collective to the performance in isolation.
Given the MLRU value of 10 Mbit/s for bandwidth, the bottleneck of each server is indeed bandwidth, being the cause for the totality of rejected calls in all experiments. Figure 2.14 shows the bandwidth measurements taken when the events of the second column of Table 2.2 are registered in the experiment with a collective formed by columbia, cambridge, and ucs. The x–axis represents the time elapsed during the experiment (in minutes). The y–axis represents bandwidth in bit/s (logarithmic scale). There are two curves in each plot of Figures 2.14(a), 2.14(b), and 2.14(c). The curve on top (labeled “total bandwidth”) is the observed total current bandwidth, including primary and non-primary simultaneous delivery sessions. The curve below the “total bandwidth” curve is labeled “bandwidth for non-primary” and shows the bandwidth utilized to deliver non-primary content. The same pattern applies to all three curves corresponding to “total bandwidth” in which the bandwidth consumption is effectively controlled at 10 Mbit/s and the non-primary bandwidth at 1 Mbit/s. In the first instant of the experiment all servers are completely free which explains that the bandwidth starts increasing from zero in the initial moments taking about an hour to reach the limiting point of maximum bandwidth consumption.

2.8 Conclusion

We have analyzed the performance of resource sharing via the formation of server collectives as a means to reduce rejection rates in content distribution services. Providers can benefit by participating in collectives but should avoid situations in which their resources are overused servicing requests on the behalf of other collectives members, worsening
the delivery quality of their own content. Our analysis and simulation via fundamental queueing models yields the following results and insights:

- We modeled fixed and elastic rate transfers within collectives and compared the rejection rates and completion times of these transfers to the case where providers operate in isolation. We then used our models to determine the conditions under which a provider benefits from participating in collectives. In particular, we determine the conditions for which all participants simultaneously benefit from their participation in collectives.

- Even a small degree of heterogeneity among participants in a collective can lead to situations in which one or more providers achieve a lower rejection rate for their content by operating in isolation. An expected consequence is that such providers would refrain from participating in collectives in these unfavorable circumstances.

- In some circumstances, we observe significant reduction in rejection rates of collectives in comparison to systems in isolation. For instance, we show a 4-order-of-magnitude reduction in rejection rates when comparing an isolated system to a two-server collective. Furthermore, a 10-server collective has a 7-order-of-magnitude reduction in comparison to an isolated system. As the number of servers increases, the relative reduction of rejection rate becomes less dramatic.

- We found asymptotic results as the number of collective providers tends to $\infty$. If the factor $\rho/k$ given by the average provider intensity $\rho$ and the maximum number of concurrent sessions in the system $k$ is less than one, then the system’s rejection
rate is 0 in the limit. Otherwise, the rejection rate converges to $1 - k/\rho$.

- When demands on providers’ contents are high, composing a collective (without thresholding) can reduce the rejection rate of all participants for a greater variation in intensities among participating systems supporting fixed-rate transfers than can be tolerated within systems supporting elastic-rate transfers.

- We analyzed three thresholding techniques that enable heterogeneous sets of server systems (different intensities and numbers of slots) to form a collective in which requests for all participants’ commodities are dropped at a rate lower than when the systems operate in isolation. We show that, in conjunction with thresholding, the ability to dynamically swap a transmission to the server that profits directly from the servicing of the content has little impact on the rejection rate. Thresholding therefore encourages providers to participate in collectives who otherwise would not do so, extending the range of heterogeneity in providers for which server collectives are applicable.

- We described the implementation of collectives for a streaming media service as a proof-of-concept that resource pooling and thresholding can be effective to reduce rejection rates at streaming media servers.
Chapter 3

Load Sharing in Services with Consistency Requirements

In the previous chapter, we analyze collectives in which every content has been replicated across a server collective once. That model accurately captures performance of delivery of static content in steady-state.

In the second part of the thesis, we turn our attention to services in which state can change as often as on a per-request basis. We develop a model that describe loads when service can be performed at a choice of multiple servers, despite a cost due to maintaining consistency on state changes. Our model is instrumental to investigate load sharing on services such as online auctions.

3.1 Overview

The success of various online services such as auctioning systems (e.g, eBay) and e-Commerce systems (e.g, Amazon.com) hinges on their ability to respond quickly to user requests, regardless of demand. Hence, the providers of these kinds of services must provision their servers to cope with intensive demands, and by consequence, to deliver
their users small response times.

Foremost, the total demand at a serving system is comprised of an aggregate of demands for multiple *servicing instances*. Example of servicing instances are auctions for an auction site, each of which brings a demand to the website hosting it. The simplest approach is to distribute requests for servicing instances such that instances are each served by a single server. Instance demands, however, fluctuate over time, thus, at moments of peak demands, response times increase at the more heavily loaded servers.

Less intense demands are placed on servers by applying *load sharing*, defined as the policy that permits requests for any highly-demanding instance to be distributed among multiple servers. As a result, the highly-demanding instances are on average served faster. An instance’s data, however, must be updated across all replications in order to preserve consistency, when multiple servers service that instance. For example, an auction site keeps updated the auctions’ current bids, hence it maintains consistency across all sets of servers serving a same auction. If one replicates instances over an arbitrarily large number of servers the cost of maintaining consistency can outweigh the savings gained from distributing the load of requests. The key solution is to find the replication factor for each of the instances, that results in lightly-loaded servers.

In actual servicing systems, server infrastructures are comprised of tens to tens of thousands of servers (see [15, 47, 53]). Load sharing is indeed applied to these server infrastructures, although its application, to the best of our knowledge, is *ad hoc*. Hence, the question as to how one should assign servicing instances to servers to reduce response times deserves attention. The fundamental questions that we address in this work are:
• How can load sharing reduce the response times for services which, when distributed across multiple servers, must be kept consistent?

• Can load sharing reduce response times when instance’s demands are already distributed such that loads across servers are approximately equal? And, if it can, what should be taken into account to determine whether load sharing should be applied in that case?

We study simple, greedy algorithms designed to decide how load should be “shared” in a serving system where there is a need to maintain consistency between the distributed instances and maintaining this consistency itself has a cost. Our work, as a contribution, models such servicing system and provides an analysis of the intensity at the busiest server obtained using such greedy algorithms compared to the configuration that would minimize this intensity (an ideal, optimal result). The outcome from minimizing the largest intensity should be a configuration in which servers' loads are within small variations, such as having all servers with average of instances' demands plus a small additional load due to consistency costs. Our analysis shows that the output given by greedy algorithms can be within a constant factor of the optimal. We investigate a number of cases that show that these algorithms produce configurations in which the maximum intensity is close to average demand of instances, usually significantly smaller than the maximum intensity in a configuration without applying load sharing, when each instance is served by a single server. This is especially important because the problem is shown to be NP-hard.

Next, we extend our model for the case in which fluctuations of instances' demands occur in small time scales. Since configurations that result in small variations of load
across servers are desired, an intuitive solution for a set of instances whose demands are approximately equal is exactly having each server to serve by itself an equal number of instances, without applying load sharing, and resulting indeed in a configuration with small variations across servers. This might suggest that no gains are obtained by applying load sharing, since additional consistency costs are then incurred. *We, however, demonstrate that when the fluctuation of demands across servers is asynchronous, response times are made smaller by applying load sharing.*

The remainder of the chapter is structured as follows. In Section 3.2, we overview related work. In Section 3.3, we describe our model for servicing systems in which demands fluctuate at large time scales. In Section 3.4, we formally define two greedy algorithms and analyze their performance. In Section 3.5, we describe an extension of the model that takes into account fluctuations of demands that occur in small time scales. We conclude the chapter in Section 3.6.

### 3.2 Related Work

In the classical scheduling problem [50] the objective is to minimize the completion time for running a set of tasks whose running times are known by distributing them over a set of machines. Such problem is related to our problem, by mapping loads of servicing instances to running times and machines to servers.  

3.1 In many systems, however, tasks cannot be fragmented, or if they can, there is not a cost for fragmenting them. Due to this reason, the algorithmic approaches for task scheduling across machines in [50, 64, 2, 95],

---

3.1 Our problem is also related to bin packing, since scheduling is related to bin packing [31].
for provisioning server loads in [1], and for traffic engineering using interior gateway routing protocols in [98] do not consider fragmentation costs in their models. Also, the optimization approaches in [120, 121, 11] and the performance study in [38] differ from our work in this same fundamental aspect.

The goals and assumptions in previous work such as in [61, 77, 56, 87, 32] that study systems for which consistency becomes a factor to be taken into account when provisioning serving resources differ from ours. The work in [61] describes the problem of minimizing the number of copies in a caching system. Coffman et al. in [32] study the number of replications of database entries to maximize the fraction of server processing needed to service only queries as a function of the arrival rate of database updates. Our goal to minimize the maximum intensity at servers is then clearly different than the goals in [61] and [32]. The works in [77, 56, 87, 32] present queueing-theoretic models to describe the dependencies in a set of database queries by the constraints that a query must wait for write tasks to be finished. In our work consistency is taken into account without focusing on timing issues such as this waiting constraint for database queries. Because of these assumptions, these models do not suit our purposes here.

The fork/join model is utilized in early studies [46, 13, 12, 106, 63, 78] to model storage systems with parallel access. Other works using a similar model focus on diverse aspects of architecture of storage systems. The works by Merchant and Yu study techniques to distribute load over multiple disk accesses by disk striping in [74] and clustered RAIDs in [75]. Also, Chen and Towsley present an evaluation for a RAID1 (mirrored disk arrays) in [27]. Eager et al. in [40] analyze for parallel processing the tradeoff in
speedup by distributing subtasks to parallel machines and the idle time generated due to contention mechanisms. Such models do not map to our problem due to particular details to be captured in these architectures, but more importantly have a distinct goal, i.e. an accurate estimation of response time given a very strict sequencing of tasks and in our work tasks are independent.

Other studies, in [30, 41, 93] present tradeoffs involved in replication of databases with models in which the goal of replication is availability. Kumar and Segev in [62] present a problem in which the goal is availability, however, limiting the communication costs. These assumptions and the goal differ from our assumptions and our goal. Yu et al. in [119] and Amza et al. in [8] focus on concurrency control which differs from our goal and can be considered an orthogonal issue. Further description on work on the performance of distributed databases is found in the survey [79] by Nicola and Jarke.

Current provisioning experiences for representative large-scale services are reported in [15, 47, 53] that describe server architectures of commercial services employing from hundreds to thousands of servers for applications such as search engines (Google [15] and Hotbot [47]) or online massive games [53]. Load sharing, among other features, is then cited for scalability, but none of these works evaluates its impact.

Finally, the studies in [45] and [17] propose to configure online games using new service paradigms such as publish/subscribe. The work in [24, 25] shows implementation of an event notification system under such paradigm. These studies, however, show how the paradigm can be applied to advance modern applications such as online gaming, without evaluating the performance of such systems. The work by Ge et al. [48] models pub-
lish/subscribe systems in order to find the intermediary node responsible for splitting the outgoing traffic that minimizes the required bandwidth. Their model does not capture the dynamics intended in our study of replication/grouping of servers. These works are not closely related to our work, except for the fact that we view our work as an important contribution for provisioning publish/subscribe systems.

The issue on how to maintain consistency is not object of study here, but an orthogonal issue. For more information on this particular issue, please refer to [37, 102, 33].

3.3 Large–Scale Services with Consistency Requirements: model and objectives

Provisioning services whose data can be updated by users poses a tradeoff in decisions on how to distribute servicing across services’ infrastructure in order to have response times reduced. In short, our goal is to investigate an alternative to the provisioning policy in which any request can only be serviced by a single server. The alternative is to allow a more flexible approach in which multiple servers are available to choose from to service a request. In this section, we consider that service demands fluctuate at rates smaller than the measurement rate used by the service provider to give estimates on service demands. A motivation for this model comes from the fact that service demand fluctuates requiring the “demand distribution” across servers to be performed periodically. Using this model measurements represent “demand snapshots” in time, which are accurate estimations of demands for every measurement interval since we do not expect great variations of demand within a measurement interval. For instance, daily fluctuations that correlate to
routine daily activities are well known [72] (e.g., consistent peaks at evenings) and typically a peak period may last for a long time. Another instance, for a ticketing system demand for booking tickets can be quite high for an hour and slow down afterwards. In Section 3.5, we present an extension of the model that deals with the case that demand fluctuates significantly within a measurement interval.

Next, we describe our model in detail.

Table 3.1: Main variables and main parameters of the model

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>Set of $m$ servers ${s_1, s_2, ..., s_m}$</td>
</tr>
<tr>
<td>$A$</td>
<td>Set of $m$ instances ${a_1, a_2, ..., a_m}$</td>
</tr>
<tr>
<td>$\gamma(a_i)$, $\gamma_i$</td>
<td>read intensity due to item $a_i \in A$</td>
</tr>
<tr>
<td>$\rho(s_j)$, $\rho_j$</td>
<td>intensity observed at server $s_j$, $1 \leq j \leq m$</td>
</tr>
<tr>
<td>$f_i$</td>
<td>fraction of intensity due to write intensity</td>
</tr>
<tr>
<td>$v_i$</td>
<td>write intensity added for placing a fragment of an instance $a_i$</td>
</tr>
<tr>
<td>$\omega_i$</td>
<td>total intensity due to item $a_i$, at any server</td>
</tr>
<tr>
<td>$w_j$</td>
<td>sum of read intensities (fragments of instance's demand) at a server $s_j$</td>
</tr>
<tr>
<td>$h_j$</td>
<td>sum of write intensities at server $s_j \in S$</td>
</tr>
<tr>
<td>$OPT$</td>
<td>algorithm that minimizes the maximum intensity</td>
</tr>
<tr>
<td>$\rho_A$</td>
<td>maximum intensity given by algorithm $A$</td>
</tr>
<tr>
<td>$\rho_{AVG}$</td>
<td>average intensity across the set of servers</td>
</tr>
<tr>
<td>$T(s)$</td>
<td>response time at server $s$</td>
</tr>
<tr>
<td>$T_{avg}$</td>
<td>average intensity across all servers</td>
</tr>
</tbody>
</table>

3.3.1 Modeling a servicing system

The provider has at its disposal a set of $m$ servers, $S = \{s_1, s_2, ..., s_m\}$, to manage the servicing of $m$ servicing instances in the set $A = \{a_1, a_2, ..., a_m\}$. Examples for such service instances include item auctions (auctioning system), multiple online games, or item sales (e-commerce).
Requests for instances are classified into two classes of requests. A first class of requests, \textit{read} requests, consists of requests that only retrieve content without altering it. In the online auction example, a read request can be a request for a description of an auctioning item. A second class of requests, \textit{write} requests, consists of requests that can alter the content being served. Examples are a seller submitting new information on an auctioning item, or a buyer submitting a bid. Each instance's demand can be described by an intensity \( \gamma(a_i) = \gamma_i \) due to read requests plus an intensity \( v(a_i) = v_i = f_i \gamma(a_i) \), \( 1 \leq i \leq l \), due to write requests. \(^{3,2}\) An instance \( a \) modified by a write request must be modified at any server servicing that instance. Therefore we define \textit{consistency requirements} to be the requirements such that instances must be kept consistent, i.e. for any \( a_i \) such that \( v_i > 0 \) there is a consistency requirement for \( a_i \).

In order to service each instance \( a \in \mathcal{A} \), two decisions are taken. A first decision concerns the \textit{fragmentation} of an instance \( a_i \)'s demand into \( z(a_i) = z_i \) fragments, \( 1 \leq z_i \leq m_i \), of that instance. The idea behind distributing pieces of an instance's intensities is to redirect requests of an instance \( a \) to the servers that serve that instance. The consequence is to divide the rate of read requests among these servers. A second decision is to assign servicing of instances or fragments of instances to servers. The \textit{load sharing} policy is defined by permitting that requests can possibly be directed to more than one server with each request to only one server at a time. Hence, there exists a non–empty subset \( \mathcal{A}', \mathcal{A}' \subseteq \mathcal{A} \), whose instances are such that \( z(a) > 1 \), \( \forall a \in \mathcal{A}' \). By contrast, a \textit{static provisioning} policy is such that \( \forall a_i \in \mathcal{A}, z_i = 1 \).

\(^{3,2}\)We generally assume that \( 0 < f_i \leq 1 \), but this condition is not necessary.
Two basic fragmentation models are considered. A fluid fragmentation is defined when an instance's read intensity can be fragmented into any arbitrary fraction of the intensity. The insight here is to view the intensity as a fluid that can be "poured" into servers continuously to reach any level. Another practical fragmentation model is an even-size fragmentation, in which fragments are of equal sizes. Here, by sharing state across more than one server, the read requests for an instance \( a \in A \) are directed to a subset of servers such that the read intensity for that instance at each of the servers is a fraction of its total read intensity. Hence, the intensity due to instance \( a_i \)'s read requests at each of the servers is \( \gamma_i / z_i \), since \( z_i \) is the number of servers servicing instance \( a_i \) and intensities are divided into equal portions.

A server's intensity describes how much of the server's processing is required in a unit of time. This is naturally a function of the read and write intensities of instances placed on a server. The intensity \( \rho(s_j) \) at server \( s_j \) is given by the sum of fragments of read intensity assigned to \( s_j \) plus the sum of write intensities on server \( s_j \).

The fact that the number of instances is equal to the number of servers is a natural modeling for a number of instances viewed as a single instance being the one in an original instance configuration. In that case, the set of servicing instances in each of the servers collapses to a single instance in our model which justifies using the number of instances \( m \).
3.3.2 Objective: maximum intensity

We now define the response time of a server $s$ as a function $T(s)$. In particular we focus on non-linear, non-decreasing functions $T(s)$ that are function of $\rho(s)$. A processor sharing model, for instance, has been used to describe servicing of tasks in servers [58]. For a processor sharing system, the function $T(s) = \frac{1}{c - \rho(s)}$, where $c$ is a constant describing a server's capacity, is the response time function for a unit size job. The capacity of a server measures the maximum amount of work processed by the server in one unit of time. The performance metric to be assessed is the maximum response time across all servers.

Since the response time is a non-decreasing function of the intensity placed on servers, our objective is to minimize the maximum intensity across servers. The largest server intensity in the solution is then given by $\rho_{OPT}$. Formally,

$$\rho_{OPT} = \min_{s \in S} \max \rho(s)$$  \hspace{1cm} (3.1)

Furthermore, in the case of systems with heterogeneous capacities a provider must maximize the minimum remaining capacity given by the difference between the capacity of a server and its intensity. 3.3

The problem of distributing fragments onto servers in order to minimize the maximum intensity can be restricted to an instance of the scheduling problem. This restriction reveals the problem to be in the class of NP-complete problems (details in Section 3.4). Therefore we study algorithms based on simple heuristics.

3.3 Minimizing the maximum intensity also reduces the variance of intensities across servers, although minimizing variance is not the primary objective here.
Let an algorithm $A$ result in a maximum intensity $\rho_A$ across servers. Our goal is to compare the result for an algorithm $A$ compared to the optimal result $\rho_{OPT}$ given by the minimum maximum intensity. We generally establish bounds that determine how large the largest intensity obtained by $A$ is compared to $\rho_{OPT}$.

We assume, without loss of generality, that a static provisioning configuration is an input to the problem. It is useful to define the original instance of a server $s_j$ to be the instance present in that server in the static provisioning configuration. Even when already having a configuration based on load sharing a provider can still compute a new configuration based on load sharing using the original item configuration corresponding to the configuration of static provisioning as an input.

We remark that the question about how to maintain consistency, rather than the impact of a policy such as load sharing given the consistency factor, is legitimate but is not the focus of our study. Answering such question is indeed orthogonal to our work.

### 3.4 Algorithms for load-sharing

In this section we state two algorithms based on a greedy policy to reduce the intensity of the busiest servers in provisioning a service with consistency requirements. We can reduce the problem to a well-known, NP-complete problem, classical scheduling, to show that the problem is NP-hard. This fact makes the study of greedy algorithms especially relevant. Let us take a particular instance of our problem in which the configuration that minimizes the maximum intensity is such that the sum of fragment sizes of every server is equal to $\rho'$ and the intensity $\rho_{OPT}$ equal at all servers $s \in S$. If an instance admits
this configuration, then it is optimal. This is true because any fragment made smaller at
some server $s$ means that a larger fragment exists in another server other than $s$, hence the
intensity in that second server is higher than $\rho_{OPT}$. Therefore, this configuration indeed
minimizes the maximum intensity found to be $\rho_{OPT}$. This instance is also a restriction
from our problem since we know the fragmentation factor $z_i$ for all instances $a_i \in \mathcal{A}$,
instead of having to determine $z_i$, $1 \leq i \leq m$ as part of the problem. In that case, let us
place fragments of arbitrary sizes, but such that the sum of fragments at every server is $\rho'$
and the sum of $z_i$ fragment sizes of any instance $a_i$ is $\gamma_i$. The problem becomes minimiz-
ing the quantities $\tau_j = \rho(s_j) - \rho'$, which is the sum of write intensities at server $s_j \in \mathcal{S}$.
Therefore placing fragments is equivalent to adding items (fragments of instances) of
given sizes $v_i$ (write intensities) above the level $\rho'$. Thus, if we solve the problem of find-
ing the configuration that minimizes the largest sum of write intensities in every server we
have also resolved our problem. This corresponds to the scheduling problem of $\sum_{i=1}^{m} z_i$
items of size $v_j$, $1 \leq j \leq \sum_{i=1}^{m} z_i$, over a set of $m$ servers $\mathcal{S}$ which is a well-known,
NP-hard problem. Therefore, the problem here is unlikely to admit a solution from a
polynomial time algorithm, under the conjecture that $P \neq NP$. $^{34}$ It is interesting to
note that when $f_i = 0$, for any $a_i \in \mathcal{S}$, the problem becomes simple to solve. In this case,
there are no write intensities and the read intensity can be placed in arbitrary sizes such
that the average intensity at every server reaches the average level.

We proceed next with useful definitions for general algorithms and simple bounds.

$^{34}$ An argument can be constructed to parallel this instance to the bin-packing problem, another problem
that belongs to the NP-complete class.
3.4.1 Useful definitions and facts

Definition A movement from $s$ to $s'$ is the transfer of a fragment of an instance's read intensity from a server $s$ to server $s'$.

A movement from $s$ to $s'$ causes an additional write intensity at server $s'$ equal to the write intensity of the instance's passed from server $s$, as previously explained.

Definition An end configuration is said to be a definitely final configuration if no possible movement can exist from a server to another without increasing the largest intensity across servers.

In an optimal end configuration, there can be no movement of fragment from one server to another at the expense of increasing the largest intensity across all servers. Therefore, a desirable property in an algorithm is that it reaches a definitely final configuration.

Fact 3.1 For a definitely final configuration the difference between intensities of any pair of servers $s_i$ and $s_j$, $\rho(s_i) \geq \rho(s_j)$, we have

$$\rho(s_i) - \rho(s_j) \leq v_{\text{max}},$$

where $v_{\text{max}}$ is the largest fragmenting cost (largest writing intensity).

This gives important insight on why the variance of servers' intensities is reduced when applying the load sharing concept. Let us also state formally a useful observation that we often use in this work.
Fact 3.2. Let \( \rho^* = \max\{\gamma_1 + v_1, \gamma_2 + v_2, \ldots, \gamma_m + v_m\} \). We have simple bounds for \( \rho_{\text{OPT}} \) as follows.

\[
\rho^* \geq \rho_{\text{OPT}} \geq \frac{1}{m} \sum_{i=1}^{m} (\gamma_i + v_i),
\]

which simply states that the optimum lies between the average of intensities across all servicing instances and the largest server intensity.

Let us also define \( \bar{f} = \frac{\sum_{i=1}^{m} f_i \gamma_i}{\sum_{j=1}^{w_j}} \), the average of \( f_i \) weighted on \( \gamma_i \), \( 1 \leq i \leq m \). Let us have for every server \( s_j \) its intensity \( \rho(s_j) \) given by the sum \( w_j \) of fragments and the sum \( h(s_j) = h_j \) of write intensities, \( s_j \in S \).

As a result of the fact above we also find an important bound for \( \sum_{i=1}^{m} \gamma_i \):

\[
\frac{1}{m} \left( \sum_{i=1}^{m} \gamma_i + \sum_{i=1}^{m} f_i \gamma_i \right) \leq \rho_{\text{OPT}}
\]

\[
\frac{1}{m} \left( \sum_{i=1}^{m} \gamma_i + \bar{f} \sum_{i=1}^{m} \gamma_i \right) \leq \rho_{\text{OPT}}
\]

\[
\sum_{i=1}^{m} \gamma_i \leq \frac{m \rho_{\text{OPT}}}{1 + \bar{f}}.
\]

Definition. A greedy policy determines the movement of an instance's read intensity or a fragment of an instance's intensity into the server of least minimum intensity. Formally, the greedy policy specified that an instance \( a \) must be placed at \( s^* \), such that \( \rho(s^*) = \min_{s \in S} \rho(s) \). This greedy policy can be generalized for a set of \( d \) fragments of a same instance to be placed in parallel into \( d \) servers with lowest intensities.

Next we use the greedy policy to construct algorithms.
3.4.2 Description of the proposed greedy algorithms

Here we build simple heuristic-based algorithms that perform movements using the greedy policy. The key idea is to limit the number of fragments to not have an excessive total write intensity at servers.

Let us consider initially an algorithm which uses the even-size fragmentation model. Such model is useful because this kind of fragmentation can be easily implemented as a distribution action taken by a request dispatcher. The requests of an instance $a_i$ are directed to a single server at rate $1/z_i$ of the actual rate without any further complexity. As part of the algorithm a value $k$ is the maximum value that any $z_i$ can assume. In the input of the algorithm we have $\rho(s_j) = \gamma(a_j)(1 + f_j)$, $1 \leq j \leq m$. The problem with passing a fragment from a server to another in only one direction, i.e., from a server of larger intensity to a server of smaller intensity is that the obtained configuration can have the maximum intensity larger than the previous one. In this case such movement should obviously be avoided. One way to improve the result is to permit an exchange of fragments such that not only a fragment is passed from the server of largest intensity to the ones of least intensity, but to also have fragments taken from the server of largest intensity and from the ones of least intensity (in a number of up to $k - 1$) and passed one another. In this case the sum of fragment sizes generally is equal to the average of $\ell \leq k$ instances' read intensities. Pick the server with greatest intensity, say server $s_i$. Now taking a number $\ell$ of up to $k - 1$ servers of smallest intensities, we proceed to the configuration that gives the smallest maximum intensity. One fragment should stay in the current server and the other $\ell - 1$ fragments are potentially placed in the servers with
lowest intensities, i.e. according to the greedy policy. After re-computing the intensities at all other servers we pick the server with the current highest intensity. The procedure continues. In the case of \( k = 2 \), the algorithm consists of attempting a series of pairwise operations between the current server of largest intensity with the server of current smallest intensity. This is algorithm \textsc{Greedy} (\( k \)). Figure 3.1 describes the steps taken in this algorithm.

Input: Each server \( s_i \in \mathcal{S} \) serves instance \( a_i \in \mathcal{A} \)
1. Sort servers \( s_1 \) through \( s_m \) by descending order of \( \gamma(a_i) \)
2. Pick initial index \( i = 1 \)
3. Make \( \ell = 2 \)
4. Split \( a_i \)'s read intensity into \( \ell \) fragments of size \( v = \gamma(a_i)/\ell \) each;
5. Compute the intensities at all servers if a movement is to keep a fragment of \( a_i \) at server \( s_i \) and place each of the \( \ell - 1 \) pieces of \( a_i \)'s read intensity following the greedy policy (and having the exchanges amongst the servers of least intensity and \( s_1 \));
6. If after this potential movement the server with maximum intensity is still \( s_i \), increment \( \ell \) (if \( \ell = k \), execute movement and go to next step) and return to step 4; otherwise, execute the movement and proceed
7. Increment \( i \); if \( i = m + 1 \), stop, otherwise go to step 3;

Figure 3.1: \textsc{Greedy} (\( k \)) algorithm

Another algorithm is \textsc{Greedy-Fluid} which takes the fluid fragmentation model. The policy here is to equalize the intensity at a set of servers with movements always in the direction that follows from passing fragments from the server of largest intensity to the ones of least intensity. This takes advantage that fragments can be of arbitrary sizes. The idea comes from an analogy of pouring fluid from a vial to another until both of them reach equal levels of fluid. This policy as opposed to having exchanges of fragments from a server to another and the fluid fragmentation model as opposed to the even-size fragmentation model make this algorithm differ fundamentally from the previous one. It
is necessary to evaluate, as part of the algorithm, the amount of fluid $x$ that should be transferred from a server $s_i$ to the largest number of servers $s \in S' \subset S$, selected using the greedy policy, such that $s_i$'s intensity decreases the most or another server, rather than $s_i$ server, becomes the one with highest intensity. By taking advantage of the fluid model, as explained, intensities of all servers in $S'$ are equal after a movement from $s_i$ of instance $a_i$'s intensity, i.e. $\rho(s_i) = \rho(s_j), s_j \in S'$. We remark that the write intensity $f_i \gamma_i$ is added to $\rho(s_j)$, if no portion of $a_i$'s intensity is contained in server $s_j$. Next the algorithm proceeds to the server with highest intensity and repeats the procedure. Figure 3.2 describes the steps taken in the GREEDY-FLUID algorithm.

Input: Each server $s_i \in S$ serves instance $a_i \in A$
1. Sort servers $s_1$ through $s_m$ by descending order of $\gamma(a_i)$
2. Pick initial index $i = 1$;
3. Make $\ell = 2$
4. Pick $\ell - 1$ following the greedy policy; re-label servers from 1 to $\ell$ by descending order of intensity
5. Compute the portion $x$ of the intensity $\gamma_i$ such that $\gamma_i - x = \gamma_j + y_j + x$, where $y_j = \rho(s_j) - \rho(s_i)$
6. Compute intensities at each server if movement of fluid as described in the previous step is done
7. If after this potential movement the server with maximum intensity is still $s_i$, increment $\ell$ (if $\ell = m$, go to next step) and return to step 4; otherwise, execute the movement and proceed
8. Increment $i$; if $i = m + 1$, stop, otherwise go to step 3;

Figure 3.2: GREEDY-FLUID algorithm

Figure 3.3 illustrates an example in which movements are performed across a set of servers $\{s_1, s_2, s_3\}$ for a set of instances $\{a_1, a_2, a_3\}$ under both fragmentation models. The bars' heights indicate the total intensity at each server. Shadowed rectangles indicate the portion of the intensity due to write requests, whereas white rectangles indicate the
portion of intensity due to read requests. The configuration on left-hand side shows the original distribution in which no load sharing is performed. The configuration in the center shows the result after the two-way movements among servers $s_1$ and $s_2$ of fragments of sizes $\gamma(a_1)/2$ and $\gamma(a_3)/2$ each. The configuration in the right-hand side shows the result after a fragment of $a_1$ of size such that the intensities at both servers $s_1$ and $s_3$ are equal. Note that on both movements the cost of write requests is added to server $s_3$. Nevertheless, the maximum intensity is reduced from the original configuration in both cases.

In particular, for the server of largest intensity $\rho_G$ for GREEDY ($k$) ($\rho_{GF}$ for GREEDY-FLUID) we can use for convenience the notation $w_G$ ($w_{GF}$) for the sum of fragment sizes and $h_G$ ($h_{GF}$) for the sum of write intensities.

Finally, the complexity of GREEDY ($k$) and GREEDY-FLUID is of the order of $O(m \log(m))$ since the algorithms require sorting instance’s intensities.

Next, we study the worst case analysis of the maximum intensity provided by these
algorithms, i.e., how bad they can perform compared to an algorithm that minimizes the maximum intensity.

3.4.3 Analysis of GREEDY (k)

**Lemma 3.3** In GREEDY (k) the total write intensity \( \sum_{s \in S} h(s) \) placed on servers is bounded as follows

\[
\sum_{s \in S} h(s) \leq \frac{mk\rho_{OPT}\bar{f}}{1 + \bar{f}}.
\]  

**Proof** Here the fragmentation of load from any instance, say \( a_i \), generates at most \( k - 1 \) fragments to be placed in servers other than server \( s_i \). In order to have an upper bound on the total cost due to write intensities in the output of the algorithm, the \( k - 1 \) write intensities should be counted for all \( m \) instances plus the write intensities that the system contains in static provisioning. Hence, \( \sum_{s \in S} h(s) \leq k \sum_{i=1}^{m} v_i = k \sum_{i=1}^{m} f_i \gamma_i \). We use then (3.3), to find (3.4):

\[
\sum_{s \in S} h(s) \leq k \sum_{i=1}^{m} f_i \gamma_i = k\bar{f} \sum_{i=1}^{m} \gamma_i \leq \frac{mk\rho_{OPT}\bar{f}}{1 + \bar{f}}.
\]

**Theorem 3.4** Let \( \rho_G \) be the maximum intensity obtained under the GREEDY (k) algorithm. Then we have the following two bounds. First,

\[
\rho_G \leq \rho_{OPT} \left(1 + 2k\frac{\bar{f}}{1 + \bar{f}}\right) = \rho_{GB2}.
\]
and, second, another bound

\[ \rho_G \leq \rho_{OPT} \left( 2 + \frac{(k - 1}\tilde{f}}{1 + \tilde{f}} - \frac{1}{m} \right) = \rho_{GB1}. \]  

(3.6)

Hence, \( \rho_G \leq \min(\rho_{GB1}, \rho_{GB2}) \).

Proof We start proving (3.5). We use the fact that in the end configuration we cannot have any further movement for any \( l \) up to \( k \). Therefore, for any number \( l - 1 \) of servers, \( 1 \leq l \leq k \), an exchange with the one with largest intensity would take the read intensities at each of the \( l \) servers to a fraction of \( 1/l \) of the read intensities, i.e. an average, but write intensities would compound in manner that can exceed the largest intensity \( \rho_G \). If such condition holds for any group of \( l - 1 \) servers to be grouped with the server of intensity \( \rho_G \), then no movement can be performed. The condition for not having any movement is formally stated

\[ w_G + h_G \leq \sum_{t=1}^{l-1} \left( \frac{w_t}{l} + h_t \right) + \frac{w_G}{l} + h_G, \]

\[ \forall \{s_t\}_{\rho(s_t) < \rho_G} \subset S, |\{s_t\}| = l - 1, 2 \leq l \leq k. \]

Now let us take \( [(m - 1)/(l - 1)] \) groups of \( l - 1 \) servers each plus the server of largest intensity. We rewrite the previous inequality and subsequently make an algebraic
manipulation to derive the following:

\[ w_G + h_G \leq \sum_{\ell=1+g(l-1)}^{(g+1)(l-1)} \left( \frac{w_\ell}{l} + h_\ell \right) + w_G/l + h_G, \]
\[ \forall s_j \in S, 2 \leq l \leq k, \]

\[ \frac{(l-1)w_G}{l} \leq \sum_{\ell=1+g(l-1)}^{(g+1)(l-1)} \left( \frac{w_\ell}{l} + h_\ell \right), \]
\[ \forall s_j \in S, 2 \leq l \leq k, \]

\[ 1 \leq g \leq \lfloor (m-1)/(l-1) \rfloor. \]

The quantities of \( w_\ell \) and \( h_\ell \) are unknown, but if we sum the read intensities placed on all servers it must equal the total read intensities over all instances. We know that the total write intensity at servers is not necessarily equal to the sum of write intensities over all instances, but certainly larger or equal to the sum of intensities over all instances and bounded as stated in Lemma 3.3. Therefore, summing up over all groups we also sum up the unknown variables \( w_\ell \) into the sum of read intensities and sum of write intensities. Formally,

\[ \frac{mw_G}{l} \leq \frac{1}{l} \sum_{i=1}^{m} \gamma_i + \sum_{g=1}^{\lfloor (m-1)/(l-1) \rfloor} \sum_{\ell=1+(g-1)l}^{g(l-1)} h_\ell \]

And more simply, \( mw_G \leq \sum_{i=1}^{m} \gamma_i + l \sum_{s \in S} h(s) \). This inequality is valid for \( 2 \leq l \leq k \). Therefore, among all values that \( l \) can assume, \( l = 2 \) provides the tightest bound for \( w_G \). Next, we simply use the bounds for both \( \sum_{s \in S} h(s) \) and \( \rho_{OPT} \) and algebraic
manipulations for a bound for $w_G$:

\[ mw_G \leq \sum_{i=1}^{m} \gamma_i + 2k \sum_{i=1}^{m} f_i \gamma_i \]

\[ mw_G \leq \sum_{i=1}^{m} (\gamma_i + f_i \gamma_i) + (2k - 1) \sum_{i=1}^{m} f_i \gamma_i \]

\[ mw_G \leq m \rho_{OPT} + (2k - 1) \frac{\hat{f} m}{1 + \hat{f}} \rho_{OPT} \]

\[ w_G \leq \rho_{OPT} \left( 1 + (2k - 1) \frac{\hat{f}}{1 + \hat{f}} \right) \]

The inequality above permits us to finish proving (3.5). For $\rho_G = w_G + h_G$, we find

\[ \rho_G \leq w_G + \sum_{i=1}^{m} \hat{f}_i \gamma_i \leq \rho_{OPT} \left( 1 + 2k \frac{\hat{f}}{1 + \hat{f}} \right) . \]

We proceed with proving the bound in (3.6). We utilize an argument very similar to the one formulated in [50]. Let the algorithm end with maximum intensity $\rho_G$. Considering that the last fragment adds a portion of intensity $y = x + v$ ($x$, read intensity and $v$, write intensity) added to a server we have that all servers’ intensities are at least $\rho_G - y$. Otherwise the last fragment should go to the server with intensity less than $\rho_G - y$. Hence, if we sum up intensities over all servers and discount the total write intensities, the result should be below the sum of read intensities over all instances. Hence we have

\[ (\rho_G - y) m + y - \sum_{s \in S} h(s) \leq \sum_{i=1}^{m} \gamma_i \]

\[ (\rho_G - y) m + y - \sum_{s \in S} h(s) + \sum_{i=1}^{m} f_i \gamma_i \leq m \rho_{OPT} \]

(3.7)
Using the bound for $\sum_{i=1}^{m} \gamma_i$ from (3.3) and the result from (3.4), we find

$$(\rho_G - y)m + y - \frac{\bar{f}m(k - 1)\rho_{OPT}}{1 + \bar{f}} \leq m\rho_{OPT} \quad (3.8)$$

Finally, using the fact that $y \leq \rho_{OPT}$ into (3.8), we prove the bound in (3.6) and finish the proof of the theorem.

$$\rho_G \leq \rho_{OPT} \left(2 + \frac{(k - 1)\bar{f}}{1 + \bar{f}} - \frac{1}{m}\right) \quad (3.9)$$

In the case of $k = 1$, which means no fragmentation whatsoever, we have the familiar result for a greedy algorithm to the classical scheduling problem, stating $\rho_G \leq \rho_{OPT}(2 - 1/m)$.

### 3.4.4 Analysis of GREEDY-FLUID

Let us first define the maximum fragmentation cost $v_{\text{max}} = \max_{a_i \in A} v_i$. In particular, if $\forall a_i \in A, f_i = \bar{f}$, then $v_{\text{max}}$ corresponds to the cost of fragmenting the instance of largest intensity defined by $\gamma_{\text{max}}, v_{\text{max}} = \bar{f}\gamma_{\text{max}}$.

**Lemma 3.5** For GREEDY-FLUID the total sum of write intensities is bounded as follows

$$\sum_{s \in S} h(s) \leq \sum_{i=1}^{m} f_i\gamma_i + (m - 1)v_{\text{max}}. \quad (3.10)$$

**Proof** Here in step 7 (Figure 3.2) we have that fluid is poured onto $k^{(1)} \leq k$ servers, i.e. $k^{(\ell)}$ denotes the number of pourings in the $\ell$ iteration. Similarly, let $\rho_G^{(\ell)}$ mean the largest
intensity in the \( \ell \) iteration. The first iteration lets \( k^{(1)} \) to have equal intensities, thus up to \( m - 1 - k^{(1)} \) servers' intensities can be above \( \rho_1 \). Hence, when step 7 is repeated fluid is poured onto \( k^{(2)} \) servers, \( k^{(2)} \leq m - 1 - k^{(1)} \). When no more movements can be executed, we have \( \sum_{i=1}^{\Delta} k^{(i)} = m - 1 \), where \( \Delta \) is the total number of rounds of movements. Therefore we have movement of fragments causing additional write intensities of at most \( m - 1 \) times and of the order smaller than or equal to \( (m - 1)v_{\text{max}} \). Finally, taking into account the initial sum of write intensities over all instances, \( \sum_{i=1}^{m} f_i \gamma_i \), we have proven the lemma.

**Theorem 3.6** For GREEDY-FLUID the maximum intensity \( \rho_{GF} \) obtained is bounded as follows

\[
\rho_{GF} \leq \rho_{GB} = \rho_{OPT} + (2 - \frac{1}{m})v_{\text{max}}.
\]  

(3.11)

**Proof** The proof starts with an argument similar to the one used to prove Theorem 3.4. We know that no movement can be performed in a very strict sense, i.e. if for a fragment of negligible size \( \varepsilon \) to be passed from the server of largest intensity to a server of smaller intensity the write intensity to be added to the one with smaller intensity causes the intensity to be larger than previously. Hence, we can use the formal statement of Fact 3.1, i.e. \( \rho_{GF} - \rho_j \leq v_{\text{max}}, \forall s_j \in S \). We then sum up over all servers \( s_j \in S \), such that we can relate the total read intensities and write intensities by the sum of read intensities over all instances and the bound found in Lemma 3.5, resulting,

\[
m\rho_{GF} - \sum_{j=1}^{m} \rho_j = m\rho_{GF} - (\sum_{i=1}^{m} \gamma_i + \sum_{j=1}^{m} h_j) \leq mv_{\text{max}}
\]  

(3.12)
Using the bound in (3.10) and manipulating the expression, we derive

\[ m \rho_{GF} - \sum_{i=1}^{j} (\gamma_i + f_i \gamma_i) - (m - 1)v_{\text{max}} \leq m \rho_{GF} - \sum_{j=1}^{m} \rho_j \]

Finally, this permits us to find \( \rho_{GF} \leq \rho_{OPT} + v_{\text{max}} \left( 2 - \frac{1}{m} \right) \), which concludes the theorem proof.

The importance of this bound is that \( \rho_{OPT} \) is without any multiplicative factor and the bound does not increase as \( m \) increases.

**Corollary 3.7** When \( f = 0 \), \( \rho_{GF} = \rho_{OPT} \), which indeed is an expected result.

### 3.4.5 Quantitative analysis

In this section we present various inputs and outputs for a service described by \( m = 16 \) instances and servers. We use as inputs the instances' intensities to be samples of either of three well-defined curves, a gaussian curve, a fast-decaying curve and a line. First, the intensity described by a gaussian function is \( \Gamma \frac{e^{-\frac{(x-\gamma)^2}{2\sigma^2}}}{\sigma \sqrt{2\pi}} \), where \( x \) is equal to the server label \( i \) for any \( s_i \in S \), \( \Gamma \), \( \gamma \) and \( \sigma \) are constants. The fast-decaying curve and the line are described by \( \Gamma \frac{1}{x^\alpha - 1} \) and \( \Gamma \frac{1}{x^m} \), where \( \alpha \) and \( \Gamma \) are constants and \( x \) is equal to the server label \( i \) for a server \( s_i \in S \). The server labels are given by an integer sequence from 1 to \( m \).

The question on how to depict the bounds is not simple to answer since it is difficult to find the optimal configuration, hence finding the largest intensity in that configuration. A monte-carlo simulation needs a large number of runs in order to produce results close to
optimal with high likelihood. In the cases considered here the number of runs necessary for accurate results is prohibitively large. We use instead a formulation in which given that the bound is a function $b = g(\rho_{OPT})$, then we depict $y = g^{-1}(\rho)$, where $\rho$ is the largest intensity in an output and $g^{-1}$ the inverse function. We always have that $\rho_{OPT} \geq y$, hence $y$ is a lower bound of $\rho_{OPT}$. If $y$ is close to the largest intensity found in an output, the bound is tight.

The notation for graphs in this Section is as follows. The server intensities are shown by the bar heights, whereas the average intensity of servicing instances and the inverse of the bound are shown by the solid and dotted lines, respectively. Intensity values are shown along the $y$-axis. Server labels are shown along the $x$-axis.

In Figure 3.4 we observe the outputs using these kinds of curves. The set of three plots in Figures 3.4(a), 3.4(d), and 3.4(g) depict each of the three inputs as described. Each of these configurations correspond to not applying load-sharing, but having a static provisioning instead. In Figures 3.4(b) and 3.4(c), we show the results after algorithms GREEDY ($k$) and GREEDY-FLUID, respectively, for a set of 16 servers in which their loads (instances) are equally spaced across a gaussian curve. In Figures 3.4(e) and 3.4(f), we show the results after algorithms GREEDY ($k$) and GREEDY-FLUID, respectively, for a set of 16 servers in which their loads (instances) are equally spaced across the fast-decaying curve as described. In Figures 3.4(h) and 3.4(i), we show the results after algorithms GREEDY ($k$) and GREEDY-FLUID, respectively, for a set of 16 servers in which their loads (instances) are equally spaced across a line also as described. In all cases, $f_i = 0.1$, for all $a_i \in \mathcal{A}$. We observe that in all outputs the maximum intensity is reduced
Figure 3.4: Results obtained using GREEDY (k) and GREEDY-FLUID for gaussian, fast-decaying, and linear inputs, \( f_i = 0.1, 1 \leq i \leq m = 16 \). Note: a re-labeling of servers is done during both procedures.
to approximately the average value, hence indicating to be close to an optimal value. The largest intensities found in the outputs (of GREEDY (k) of Figures 3.4(b), 3.4(e), 3.4(h) are a reduction to about 63%, 50%, and 54% of the largest intensities in each respective inputs. The same applies for the outputs of the GREEDY-FLUID algorithm in Figures 3.4(c), 3.4(f), 3.4(i). In the case of the outputs from GREEDY-FLUID the bounds are close to the largest intensity obtained in the outputs as expected.

Another type of input is obtained from selecting $m$ random numbers between 0 and $m$, typically a sorted sequence $\{x_i\}$, $x_i = U(0, m)$, $1 \leq i \leq m$, $U$ a generator that outputs uniformly distributed numbers between 0 and $m$ and applying the functions described previously to each of the $x$ values in the sequence $\{x_i\}$. As a consequence, the read intensities of the service's instances are also randomized.

Figure 3.5 depicts results obtained when randomizing the inputs as described. We show in Figures 3.5(a), 3.5(d), 3.5(g) a set of three plots which depict the inputs for the randomized gaussian, fast-decaying and linear-increasing inputs, respectively. In Figures 3.5(b) and 3.5(c) the largest intensity in both outputs (of GREEDY (k) and GREEDY-FLUID, respectively) is reduced to about 63% of the largest intensity in the inputs. In the case of outputs (of GREEDY (k) and GREEDY-FLUID, respectively) shown in Figures 3.5(e), 3.5(f), for the fast-decaying curve, and also outputs (of GREEDY (k) and GREEDY-FLUID, respectively) shown in Figures 3.5(h) and 3.5(i), similar comparisons indicate a reduction of the largest intensities in each of the cases to about 75% and 58%, respectively, of the largest intensity in the respective inputs.

Next we observe in Figure 3.6 inputs and sets of results for the cases where $\tilde{f}_i = .2$
Figure 3.5: Results obtained using GREEDY (k) and GREEDY-FLUID for gaussian, fast-decaying, and linear inputs (all randomized), $f_i = .1, 1 \leq i \leq m = 16$. Note: a re-labeling of servers is done during both procedures.
Figure 3.6: Results obtained using GREEDY ($k$) and GREEDY-FLUID for fast-decaying input function, $f = .2$ in (a), (b), (c), $f = .3$ in (d), (e), (f), $m = 16$. Note: a re-labeling of servers is done during both procedures.
in Figures 3.6(a), 3.6(b), and 3.6(c) and \( f_i = .3 \) in Figures 3.6(d), 3.6(e), and 3.6(f), \( 1 \leq i \leq m \), for the fast-decaying curve as input. In this case the values of \( f_i \) are higher than those in Figure 3.4. In Figure 3.6(b) the largest intensity output from GREEDY \((k)\) is 62\% of the largest intensity in the input. For the output from GREEDY-FLUID in Figure 3.6(c) the largest intensity is 54\% of the largest intensity in the input. In Figure 3.6(e) the largest intensity output from GREEDY \((k)\) is 67\% of the largest intensity in the input, whereas for GREEDY-FLUID (Figure 3.6(f)) it is 57\%. This results show a more significant difference between the outputs of GREEDY \((k)\) and GREEDY-FLUID as \( f = f_i \) increases. Here, the bound for GREEDY \((k)\) is reasonably close to the largest intensity found in the output, but for GREEDY-FLUID the bound is worse, which is a result of having \( v_{\text{max}} \) larger as result of values of \( f_i, 1 \leq i \leq m \), also larger.

![Graphs](image)

**Figure 3.7:** Results obtained using GREEDY \((k)\) and GREEDY-FLUID for linear input, \( f \) described by a linear, decreasing function, \( m = 16 \). Note: a re-labeling of servers is done during both procedures.

We also study inputs and outputs of fast-decaying and gaussian inputs where \( f_i \) is not equal for all servers but a decreasing linear function of the server label, described by

\[ f_i = 0.2(1 - \frac{1}{m}). \]

Results are shown in Figure 3.7, where Figure 3.7(a) depicts the input
and Figures 3.7(b) and 3.7(c) depict the outputs for GREEDY (k) and GREEDY-FLUID, respectively. Here, $\bar{f} = .07$, an average weighted on the input demands, and the input demands are given by the linear increasing curve. The outputs exhibit largest intensities which are about 54% of the largest intensity in the input for Figure 3.7. The bounds indicate values close to the ones found by the largest intensity in each of the outputs.

We also study cases in which the writing intensity is equal for all instances, i.e. $v_i = f_i \gamma_i = v_j = f_j \gamma_j$ for any $a_i, a_j \in \mathcal{A}$. Therefore, in this case the fragmenting cost of an instance's read intensity is a constant, regardless of the instance. We show in Figure 3.8 an example of input given by a gaussian curve and equal write intensities. Here, Figure 3.8(a) depicts the input, whereas Figures 3.8(b) and 3.8(c) plot the output for GREEDY (k) and GREEDY-FLUID, respectively. The outputs of both GREEDY (k) and GREEDY-FLUID show a reduction to about 63% of the largest intensity in the output from the largest intensity from the input. The bound gives a value close to the one found by the largest intensity, especially in the case of GREEDY-FLUID in Figure 3.8(c).

Figure 3.8: Results obtained using GREEDY (k) and GREEDY-FLUID for gaussian input, write intensities $f_i \gamma_i$ equal for any $a_i \in \mathcal{A}$, $m = 16$. Note: a re-labeling of servers is done during both procedures.
Figure 3.9: Results obtained using GREEDY (k) and GREEDY-FLUID for fast decaying input, \( f_i = .1 \) for any \( a_i \in A, m = 1000 \). Note: a re-labeling of servers is done during both procedures.

We present the last set of results in this section using a large number of instances and servers. Here \( m = 1000 \). In Figure 3.9(a), we show an input in which the intensities are given by a fast-decaying curve when \( m = 1000 \). In Figures 3.9(b) and 3.9(c), we show the outcomes of GREEDY (k) and GREEDY-FLUID, respectively. The server labels are still shown along the x-axis and intensity values shown along the y-axis, but we show the intensities in the input and the results via curves, instead of bars. Similar to the previous results, the intensities in the algorithms’ outcome is not continuous, since the server labels form a discrete set of integers from 1 to 1000. Because of the density of discrete points, however, the curve appears as a continuous curve. In Figure 3.10(a), we show an input in which the intensities are given by a gaussian curve when \( m = 1000 \). In Figures 3.10(b) and 3.10(c), we show the outcomes of GREEDY (k) and GREEDY-FLUID, respectively. The conclusions drawn from these results are similar to the conclusions drawn previously. The intensity at the busiest server observed in the outcomes of the algorithms is much closer to the average intensity of instances than the largest intensity in the input.
Figure 3.10: Results obtained using GREEDY(k) and GREEDY-FLUID for gaussian input, $f_t = .1$ for any $a_i \in A$, $m = 1000$. Note: a re-labeling of servers is done during both procedures.

### 3.5 Fast fluctuation of demands

In the previous section we examine simple algorithms that can be applied to distribute servicing across a set of servers and to reduce intensity and consequently response time. It is then assumed that averages of instances’ intensities over measurement intervals give an accurate description of the system. These averages are snapshots taken over measurement intervals. But in some systems demands can fluctuate at a rate larger than the rate at which measurements are taken. Furthermore, the measurement rate can be constrained by practical limitations, thus cannot be made as small as desired. Examples include any instance of flash demand, i.e., demands that peak up unexpectedly and go down in a small timescale. For instance, activity of an auctioning system can have sudden, short-lived peaks due to a submitted bid that triggers a series of other bids from other users competing in the same auction. It is also well known that auctions’ activity increases when close to their deadlines for submitting bids. As a matter of fact, many services might exhibit both slow and fast variations. For instance an auction system might exhibit eventual peaks of
demands and slow fluctuations of demand as well.

Figure 3.11: The average between asynchronous periods of high and low intensities for two distinct servers containing two distinct instances each.

Figure 3.11 illustrates the case of two servers $s_1$ and $s_2$ each servicing instances $a_1$ and $a_2$, respectively. They exhibit equal intensity shown by the height of the bars, but the plot of $\rho(s_1)$ and $\rho(s_2)$ (y-axis in right hand side of figure) as times varies (x-axis) shows that when $a_1$ exhibit high demand, $a_2$ exhibits low demand, and vice-versa. The key idea here is to explore the possibility of statistical multiplexing across the demands of the two servers by load sharing servers $s_1$ and $s_2$.

In this case a system with servers containing equal average response times can differ greatly with fluctuation of demand. We demonstrate in Section 3.5.4 that homogeneous systems in which intensities at servers are equal on average can have response times significantly reduced if taking the fluctuation of demands *within a measurement interval* into account. This goes against the intuition that no load sharing should be performed in systems already balanced.
3.5.1 Extension of model

As previously, a service such as an auctioning system, search engine, or an e-commerce can be viewed as the servicing of a collection of instances. A provider hosts an online service servicing instances in the set $\mathcal{A} = \{a_1, a_2, \ldots, a_l\}$. This provider deploys a set of servers $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$, $l \geq m$.

Now the number of instances in a server is a relevant number to be analyzed and the modeling by aggregating instances into a single instance can result less accurate because fluctuations of demand happen within measurement intervals.

The model is extended to describe the intensity process for instances $a_1$ and $a_2$ by modulated processes each that describes demands in high and low demand periods. For an instance $a_i$ read intensities in high and low demand periods are $\gamma_i$ and $\gamma_i'$, respectively, and for write intensities in high and low demand periods $\nu_i$ and $\nu_i'$, respectively, $\forall a_i \in \mathcal{A}$. The total intensity for an instance $a_i$ in high and low demand periods is $\omega_i$ and $\omega_i'$, respectively, which correspond simply to the sum of read and write intensities, i.e., $\omega_i = \gamma_i + \nu_i$ in high-demand periods and $\omega_i' = \gamma_i' + \nu_i'$ in low-demand periods. The instance’s intensity is high with probability $p_i$, and, conversely, low with probability $1 - p_i$. Similar to the description in Section 3.3, the ratio between the write intensity $\nu_i$ and the read intensity $\gamma_i$ is $f_i = \frac{\nu_i}{\gamma_i} = \frac{\nu_i'}{\gamma_i'}$.

A single server is described by a processor sharing queue with Poisson arrivals and capacity $c$. The intensity $\rho(s)$ at a server is a random variable that is a function of the number of instances placed at $s$ and the state (high, low) for each of the instances.

The goal here is to provision servers using the load sharing concept such that the
average response time, described by the random variable $T$, is reduced. We want to compare configurations of cluster sizes $z_i$ such that the response times is reduced. As a consequence the amount of incoming requests directed at each server is also determined.

3.5.2 Analysis of the intuitive case $z = 2$

In particular we consider the case in which $\omega_1 = \omega_2 = \omega$, $\omega'_1 = \omega'_2 = \omega'$, $\gamma_1 = \gamma_2 = \gamma$, $\gamma'_1 = \gamma'_2 = \gamma'$, $v_1 = v_2 = v$, $v'_1 = v'_2 = v'$, and $p_1 = p_2 = p$. This case is interesting since it yields equal averages for both servers. Since in a configuration given by static provisioning the response times are already balanced, one might think by a simple intuition that the consistency cost increases response time when applying load sharing, but we demonstrate that this is not necessarily the case. In fact, we proceed to demonstrate in this section that applying load sharing can reduce response times.

It is convenient to define random variables $N_1$ and $N_2$, each describing if instances $a_1$ and $a_2$, respectively, are in high demand periods ($N_1 = 1$, $N_2 = 1$) or in low demand periods ($N_1 = 0$, $N_2 = 0$). It is further convenient to define the joint probability $\pi_{u,v} = P(N_1 = u, N_2 = v) = p^{u+v}(1-p)^{2-u-v}$. The probability that a random request is one of instance $a_1$ is $\beta_1 = \frac{u\omega+(1-u)\omega'}{(u+v)\omega+(2-u-v)\omega'}$, conditioned on $N_1 = u$ and $N_2 = v$. Similarly, the probability that a random request is one of instance $a_2$ is $\beta_2 = \frac{u\omega+(1-u)\omega'}{(u+v)\omega+(2-u-v)\omega'}$, conditioned on $N_1 = u$ and $N_2 = v$. The average response times for a unit-size job conditioned in the states of instances $a_1$ and $a_2$ are, respectively, $E[T|N_1 = u] = \frac{1}{e-(u\omega+(1-u)\omega')}$ and $E[T|N_2 = v] = \frac{1}{e-(u\omega+(1-v)\omega')}$. 
The static provisioning configuration yields an average response time as follows.

\[
E[T]_{z=1} = \sum_{u=0}^{1} \sum_{v=0}^{1} (\beta_1 E[T|N_1 = u] + \beta_2 E[T|N_2 = v]) \pi_{u,v}, \quad (3.13)
\]

where \( z = 1 \) says that each of the instances is served at only one server, hence the static provisioning approach. For load sharing the response time conditioned on both state variables \( N_1 = u \) and \( N_2 = v \) is \( E[T|N_1 = u, N_2 = v] = \frac{1}{c-(u+v)\omega+(1-u)(1-v)\omega'}(1-\frac{1}{2(1+f)}) \).

When \( f \) is small, say \( f = 0 \) for the extreme case, and both instances are in a high–demand period, i.e., both \( N_1 = u = 1 \) and \( N_2 = v = 1 \), the intensity is equal to the intensity in static provisioning. If only one of them is in high–demand period, however, then the intensity can be cut by a half from the factor \( 1 - \frac{1}{2(1+f)} \), which can result in significant reduction in response time. The reduction in average also occurs since the probability of only one of them at high–demand period is higher than the probability of having both at high–demand periods for \( p \) small. Finally the average response time using load sharing is derived:

\[
E[T]_{z=2} = \sum_{u=0}^{1} \sum_{v=0}^{1} E[T|N_1 = u, N_2 = v] \pi_{u,v}, \quad (3.14)
\]

where \( z = 2 \) says that each of the instances is served on both servers, hence the load sharing approach.

### 3.5.3 Generalized Analysis

We investigate, similar to the previous section, the case for which \( \omega(a_i) = \omega(a_j) = \omega \), \( \omega'(a_i) = \omega'(a_j) = \omega' \) and \( p_i = p_j, \forall i, j, 1 \leq i \leq j \leq l \), and all servers have capacity \( c \).
In strict partitioning $z = 1$, but in load sharing $z > 1$. In particular, we assume clusters, i.e. groups of servers, to serve instances, such that any two different instances can each be served by a server from respective sets that are either without any intersection or equal. In other words, the clusters form disjoints sets of servers. Let $n$ be the number $n$ of clusters.

We have that $n = m/z$. The number of instances per cluster is given by $q = lz/m$. Let us number clusters and define $N_j$ as the number of busy instances at a cluster $j$. Then the response time conditioned on variables $N_1, \ldots, N_n$ that describe the number of busy instances at each cluster of servers is as follows:

$$T = g(N_1 = i_1, \ldots, N_n = i_n) = \frac{\sum_{j=1}^{n} \rho(i_j) r(i_j)}{\sum_{k=1}^{n} \rho(i_k)},$$

where $r(i_j)$ is a function $r(i_j) = \frac{1}{(c-(i_j\omega+(lz/m-i_j)\omega')(u/z+1-u))}, \rho(i_j) = \omega i_j + (lz/m - i_j)\omega'$, and $u = \frac{1}{1+\frac{1}{2}}$. The function $\rho(i_j)$ describes the intensity at a server of cluster $j$ as a function of the number of busy instances in a server of that cluster. When $N_1 = i_1, \ldots, N_n = i_n$, then the probability that a request at random is to be served at a server of cluster $j$ is $\frac{\rho(i_j)}{\sum_{k=1}^{n} \rho(i_k)}$.

The expectation for $T$ is given by definition as follows

$$E[T] = \sum_{i_1=0}^{q} \sum_{i_2=0}^{q} \ldots \sum_{i_n=0}^{q} g(i_1, \ldots, i_n) \pi(i_1, i_2, \ldots, i_n)$$

(3.15)

where $\pi(i_1, i_2, \ldots, i_n)$ is joint density function of $N_1, \ldots, N_n$. 
3.5.4 Numerical Results

The graphs in Figure 3.12 show the response time along the y-axis, whereas the fraction $f$ of read requests in Figure 3.12(a) and the probability $p$ are shown along the x-axis in Figure 3.12(b). The scenario here is given by $\omega = 1$ in high periods, $\omega' = .1$ in low periods, $c = 2.5$, $f = .111$ (Figure 3.12(b)), and $p = .2$ (Figure 3.12(a)).

In Figure 3.12(a) the curves show the results obtained for $z = 1$, and $z = 2$, meaning that only one ($z = 1$) instance serviced per server (static provisioning) and two instances ($z = 2$) serviced at each server (load sharing). The results when varying $f$ indicate the expected tradeoff in the amount of read requests, i.e., starting from approximately $f = 0.8$ downward the response times using $z = 2$ (load sharing) are smaller than $z = 1$ (static provisioning).

Figure 3.12: The case of two servers and two instances. Here, $\omega = 1$ in high periods, $\omega' = .1$ in low periods, $c = 2.5$, $f = .111$ (in Figure 3.12(b)), and $p = .2$ (in Figure 3.12(a)).

In Figure 3.12(b), by observing the probability $p$ of an instance being on a high-
demand period such that \( p \leq 0.8 \), the response times when \( z = 2 \) are smaller than the ones when \( z = 1 \). When \( p \) is high, in this case \( p > 0.8 \), both instances are very likely to be busy simultaneously, hence load sharing does not introduce a better result than a static provisioning solution. By contrast, when \( p \leq 0.8 \), the probability that either one of the instances (but not both) being in high-demand period and the other one in a low-demand period is larger than the probability that both are on high-demand period. This fact, along with the better load distribution, explains why load sharing results in smaller average response times. The exception happens when \( p \) is low, since most likely both are on low-demand period and the probability of either one being on high-demand becomes smaller. This fact explains why in Figure 3.12(b) as \( p \) decreases from \( p = 0.4 \) downward, the gap between the two curves gets narrower.

Figure 3.13: The case of many instances but still a homogeneous system. \((m = 60, p = .1, c = 3, f = .25, \omega(a) = 1 \text{ in high periods}, \omega'(a) = .01 \text{ in low periods})\).

We investigate then for a number of instances greater than 2. Here \( m = 60 \) servers, and we vary \( l = 60, 120, 180 \). Figure 3.13 shows the results in this case. Response time
is shown along the $y$–axis, whereas the number $l$ of instances is shown along the $x$–axis. The curves $z = 1, 2, 3, 4, 5, 6$ show results obtained when using $z$ as the replication factor, i.e., $z$ servers servicing each instance. We remark that $q = lz/m$. We observe that the response time decreases as the replication factor $z$ increases up to $z = 5$. For $z = 6$ the curve approximately overlaps with the curve for $z = 3$, which results in larger response times than for $z = 4$ or $z = 5$. This exemplifies the tradeoff between load sharing and response times as discussed previously.

Here we observe the case in which servers are considered equally loaded in average and a simple intuition would lead to a conclusion that load sharing should not be performed since write requests place an additional write intensity to servers’ intensities. The counter–intuitive results, however, demonstrate that load sharing can reduce response times to approximately 75% of the response times observed when no load sharing is performed. As previously explained the variation of demands between periods of high and low intensities and an “asynchrony” between instances’ demands make possible a statistical multiplexing under a load sharing configuration.

3.5.5 Discussion

Since the random variables $N_1, ..., N_n$ are independent then the joint density function $\pi(i_1, ..., i_n)$ is given by the product of the density functions $\pi(i_1, ..., i_n) = \prod_{j=1}^n \pi(i_j)$. In the case of our problem, these density functions are given by the binomial function. But the function $g(.)$ is a function of $N_1, ..., N_n$ which makes the analysis of the expression given in (3.15) difficult to simplify. For numerical analysis, the number of all combi-
nations for $N_1, \ldots, N_n$ can be very high. This can also make the computation of $E[T]$ lengthy, depending on values of $l$, $m$, and $z$, even though we find data structures that make this procedure faster.

Such considerations demonstrate the limitations in adopting this model to find the optimal number of replications, i.e. values of $z$ that minimize the response times.

### 3.6 Conclusion

In this work we have studied the impact of load sharing in provisioning services with consistency requirements such as auctioning and e-commerce websites whose providers can employ a number of servers on the order of up to tens of thousands to serve large-scale demands. A provider must take into account that consistency requirements place an extra burden to servers that "share" load. We model such servicing systems to investigate the fundamental factors on how to apply load sharing.

The problem of minimizing the maximum intensity across servers as a way to reduce response times is found to be NP-hard. We have defined greedy algorithms for this problem and analyzed the outcomes from these greedy algorithms. The analysis shows that for the outcome of the GREEDY-FLUID algorithm, the largest intensity is always within a constant factor of the optimal. We observe that by applying load-sharing response times can be significantly reduced, since in quantitative analysis of representative cases the largest intensity can drop to approximately 50% of the largest intensity in a configuration in which all service instances are each served by a single server.

We also demonstrate that, even when response times given as result of static provi-
sioning are equal in all servers, average response times when applying load sharing are smaller, if servicing instances fluctuation of demands are asynchronous. We show, for example, a simple scenario, in which intensities of two servers are equal with each of them serving a single instance, and by having the two servers serving both instances, response times are made approximately 10% smaller. This result is insightful and we can definitely expect a greater reduction of response times for comparisons of results that consider load-sharing to those that do not for larger number of instances and larger number of servers.
Chapter 4

Third-party Resource Management of E-Commerce Websites

In the first two problems addressed in the thesis we did not assume any management entity in charge of building collectives and provisioning resources among them. In this part of the thesis we consider that a server provider, acting as a manager entity, has servers and provides them to service providers. Here, provisioning resources must result in levels of service accepted by service providers, each of which has its own service’s objectives. The server provider’s objective, however, is maximizing its profits.

In particular we focus on provisioning issues for a server provider that provisions servers to e-commerce websites. A service level agreement between a server provider and a e-commerce website owner defines quantitative requirements for the e-commerce service, typically response times in the website access. It is important for the server provider to not violate service level agreements. Avoiding such violations increases profits. We investigate methods that maximize server provider profits.
4.1 Overview

Companies that offer e-commerce applications often contract a third-party web service provider to provide and manage the necessary computing infrastructure. To be profitable, these providers simultaneously service multiple customer companies, maintaining a separate service level agreement (SLA) with each customer. The stochastic nature of request arrivals and service times makes it impossible for the provider to meet the conditions of the SLA for every request it hosts. Hence, as part of the SLA agreement, the provider is charged for each service miss: a request whose service does not meet the requirements specified in the SLA. A financially sound strategy for the provider is to provision its resources among its set of customers in such a way that its profits are maximized, which translates to minimizing the charges accrued as a result of server misses.

The web servicing architecture used by providers typically consists of three tiers, each of which is provisioned independently. The front-end serving tier handles all simple, static web transactions, composed of HTTP (HTTPS) requests. The application tier handles more complex, dynamic queries that might involve the execution of java servlets or scripts. The database tier handles requests that involve the lookup of specific, non-cached data, such as ones’ personal banking records.

The amount of work that must be performed by the servers that support the front-end tier is low enough that overprovisioning is a cheap solution to meet SLA requirements. It has been shown in McWerther et al. [70] that prioritized scheduling can improve the database tier’s ability to meet its imposed SLA requirements. With methods to maximize profits in these two tiers, a provider’s profits then depend on how well it provisions its
application tier resources to service requests to that tier. If not configured properly, a customer can suffer numerous service misses which the provider winds up paying for. Surprisingly, there has been little work that assists providers in how to configure their application tier.

Here, we explore how a provider should allocate its application tier serving resources among its set of customers with whom it has established SLAs. A desirable allocation is one that maximizes profits by minimizing the cost accrued as a result of service misses. Along this front, we make three major contributions:

• First, we identify an appropriate model for the servicing system of the application tier.

• Second, we formalize the problem of allocating serving resources to multiple customers to maximize provider profits as an optimization problem.

• Third, we derive three efficient approximation methods that allocate resources to customers, and show through simulation that the allocations produced achieve profits that are close to optimal and are significantly higher than the profits achieved via simple heuristics.

To identify an appropriate model for the application tier servicing system, we analyze an e-commerce website (department store) traces of requests for dynamic content from 2001. Using these traces, we characterize the arrival process of requests to the application tier. We find that, for the horizon time of interest, the arrival process is adequately described by a Poisson process.
Our formulation of the optimization problem models each of the provider's servers as an independent M/G/1/PS queue. The solution minimizes an objective function that describes the cost resulting from service misses. We compute allocations by deriving three approximation algorithms, each of whose computational complexity is linear in the number of customers that the service provider hosts. The first approximation assumes that the average servicing time of jobs for each customer is known. The second approximation requires both the first and second moments of the service time distribution. The third approximation method utilizes known bounds for the class of Exponential Bounded Burstiness processes, to which the Poisson process belongs. In special circumstances, a provider is able to estimate the adequate size of its total serving resources via a simple equation.

Results from experiments using event-driven simulation show that our approximation methods come close to minimizing service miss costs. We also compare the costs computed by the approximation methods to costs computed by naive heuristics. The results show that, by comparing results of our approximation methods to the ones obtained via naive heuristics, service providers can decrease their costs by 60%, hence increasing their profits by a significant margin.

This chapter is organized as follows. In Section 4.2 we overview related work. Section 4.3 describes our model of the service provider and poses the optimization problem. In Section 4.4 we demonstrate that, for timescales of interest, the arrival process of requests to the application tier is effectively Poisson. Section 4.5 describes our approximation algorithms, and Section 4.6 shows results from simulations that demonstrate the
performance of our algorithms. We discuss how the optimization framework permits to
determine the necessary number of servers in Section 4.7. We present our concluding
remarks in Section 4.8.

4.2 Related Work

Previous work that investigates e-commerce workloads differs from our characterization
study here in that it does not address the application-tier workload. Instead, studies in [73, 
76] focus on the workload at the front-end tier of a website. There has been little progress
in characterizing e-commerce website workloads imposed specifically by jobs whose
content is generated dynamically. There are studies [26, 97, 94] that investigate dynamic
content but either do not analyze server workloads or do not investigate e-commerce
sites in particular. In [94], the used approach is to have an instrumented client collecting
measurements, but parameters of relevance to our study such as rate of request arrivals at
a server are not possible to be measured because measurements occur in the client side,
rather than in the server side. In [26, 97], traces of a sporting event website are analyzed.
The characterization of these workloads can differ from e-commerce website workloads
as we show later (e.g., the distribution of time between request arrivals exhibits different
properties).

There have been numerous performance studies focusing on the problem of allocating
a fixed set of resources in order to balance the load across the set of resources. Game-
thoretical models for resource allocation have been proposed (refer to [66] and therein for
additional references) under the assumption that the resources to be pooled are controlled
by providers that do not cooperate. In contrast, our work focuses on a single provider that controls the allocation of its serving resources. Federgruen and Groenvelt [43] take an algorithmic approach to find conditions for the optimization formulation of resource allocation problem where resources are given in discrete units. This approach was subsequently generalized by Wolf and Yu [114]. Tantawi and Towsley [103] explored a similar problem via a graph-theoretic formulation, solving a resource-allocation optimization problem. These authors further extended this work in [104] to the case where there are multiple classes of resources. Our work is different in that our goal is not to balance load, but is to maximize profits. In particular, when customers' charges differ for service misses, a simple load balancing strategy is less favorable than a strategy which diverts resources to meet the load of higher-cost service misses at the expense of missing lower-cost service misses. In [91], Sairamesh et al. explore the problem of allocating network link capacity to different traffic classes in order to minimize costs. Their formulation, however, does not map easily to the provisioning problem at the application serving tier. They utilize an $M/M/1/B$ queueing model with a finite buffer of size $B$ for which the blocking probability is used as a utility function of the link capacity. Furthermore, service level agreements are usually specified as a function of response time and not as a function of the blocking probability.

A small body of work uses analytic models to evaluate e-commerce serving systems. Almeida et al. [6] propose a scheme of priority levels as a function of potential revenue estimated for different types of requests to a single website (front-end tier). The works by Liu et al. [67, 68] and Farias et al. [36] are closest to our approach; both use a general
concept of a cost model given by a product between customer request rate and fraction of requests that violate service level agreements for the front-end serving tier. There are several aspects of practical application-tier serving systems that are captured by our work that are not captured in these previous studies. For instance, each provider's server is dedicated to a particular customer, and may not be shared among multiple customers (as is assumed in previous work). Our methods permit us to find allocations via a pair of equations or, in special circumstances, a single-equation solution, both of which are simpler than the methods from [36, 67, 68]. This is of great importance in practical settings where a provider can adjust its serving infrastructure when loads fluctuate over time. Finally, in [105] a network of queues is proposed as a model for the multiple tiers in an e-commerce website, where each queue describes a tier of the system. The analysis of internals of the multiple tier architecture can extend the work done here.

4.3 Model for the Application Serving System

Our model consists of two classes of participants: a single e-business provider and \( m \) customers which we number 1 through \( m \). Each customer \( i \) individually establishes a service level agreement (SLA) with the provider, the details of which are described below. The provider has at its discretion \( n, n > m \), servers that it uses to service the requests of the \( m \) customers. As is typically done in practice, the provider selects the number of servers that it will dedicate to each customer such that no two customers' requests are serviced by the same server. Each customer, who then offers services to clients (users), can independently arrange a certain level of service to each client. Our study focuses on
the SLA between a server provider and its customers. The study of arrangements between customers and their clients lies outside the scope of this dissertation.

The SLA for each customer \( i \) includes a charge, \( p_i \), that is paid by the customer to the provider for each request the provider services. It also includes a refund \( c_i \), per request in case the service provider exceeds a response time requirement. As part of the response time requirement, the SLA defines, for each customer \( i \), a series of \( s_i \) service demand levels, where the \( k \)-th demand level has associated with it a maximum tolerated response time \( d_{i,k} \), \( 1 \leq k \leq s_i \) where \( d_{i,k} < d_{i,k+1} \) for all \( 1 \leq k < s_i \). The service demand time of an instance of a transaction for a customer is the amount of time it would take to process the transaction when a single application server is dedicated to processing only this transaction. We let \( B_i \) be a random variable that describes the service demand time of a transaction, and define \( S_i \) to be a random variable that describes demand levels from 1 to \( k \) as a function of the transaction demand \( B_i \). We describe the time that the servicing system actually takes to process a transaction of service demand, given that \( B_i = w \), as \( D_i(w) \). When more than one transaction is processed simultaneously sharing the same serving resource, the response time \( D_i \) results larger than the service demand time \( B_i \). The probability of violating the response time requirement (SLA violation) is

\[
\sum_{k=1}^{s_i} P(D_i(w) > d_{i,k}|S_i = k)P(S_i = k).
\]

We assume that all demands for customer \( i \) that fall into the same service demand level have identical service demand times, i.e., that there is a constant estimator, \( w_{i,k} \) for \( B_i \) whenever \( S_i = k \). In this case, the probability of violating the response time requirement can be rewritten as

\[
\sum_{k=1}^{s_i} P(D_i(w_{i,k}) > d_{i,k})P(S_i = k).
\]

The SLA must then also contain
a mapping from estimators \( w_{i,k} \) of a service demand level to a response time \( d_{i,k} \). In the case that \( S_i \) is a non-decreasing "quantization" function of \( B_i \), the set \( \{\ell_1, \ell_2, \ldots, \ell_k\} \) comprise the "jump points" of the quantization function such that \( S_i = k \), for \( \ell_{i,k-1} < B_i \leq \ell_{i,k} \). A possible choice for \( w_{i,k} \) is the average given by \( \int_{\ell_{k-1}}^{\ell_k} w d\tilde{B}_i(w) \), where \( \tilde{B}_i \) is the distribution function of \( B_i \). When only one level is used (\( s_i = 1 \)), then \( w_i = \int_0^\infty w d\tilde{B}_i(w) = E[B_i] \). In practice, a provider uses only one demand level, e.g. using the average service demand time as estimator. Our formulation permits a more general approach that could be used for instance when the service demand time is described by a multimodal distribution. Also, an SLA may be specified only in terms of worst-case assumptions, i.e. a maximum tolerated response time for the larger envisioned service demand expressed as a single demand level. Our model permits this type of specification. An allocation based on a worst-case assumption, however, may result in underutilized servers when compared to an allocation based on a multiple-level service demand SLA.

We let \( \Lambda_i \) be the arrival rate of requests from customer \( i \). We assume that the response time \( D_i(w_{i,k}) \), \( 1 \leq i \leq m \), are identically distributed random variables. A provider's profits can then be written as:

\[
\sum_{i=1}^{m} \Lambda_i (p_i - c_i \sum_{k=1}^{s_i} P(D_i(w_{i,k}) > d_{i,k}) P(S_i = k))
\]

and the provider's objective is to maximize this quantity. Note that the manner in which the provider chooses to provision its servers affects only \( P(D_i(w_{i,k}) > d_{i,k}) \). Hence, the provisioning problem is equivalent to one in which the goal is to provision the servers among customers such that \( \sum_{i=1}^{m} \sum_{k=1}^{s_i} \Lambda_i P(D_i(w_{i,k}) > d_{i,k}) P(S_i = k) c_i \) is minimized.
We assume that each server is work conserving, and splits its processing power evenly among all of its simultaneously active jobs. The actual time, $D_i(w)$, spent servicing the transaction depends heavily on the number of other jobs being serviced by the system. An application server typically relies on a round-robin mechanism implemented in its operating system that concurrently services multiple requests. This mechanism assures that an equal time interval ("quantum" of computation) is assigned to each simultaneous job. After this quantum of computation, the job must wait for the other concurrent jobs to each receive a quantum. This process repeats for a job until its servicing is finished. In theory, as the quantum is made very small, in the limit this servicing mechanism becomes a processor sharing system. In practice, the quanta are small enough such that a processor sharing system provides a fairly accurate model of serving systems. Also, in application servers, a server's servicing resources are finite, such that the number of jobs that can be serviced simultaneously is bounded. Since SLAs incorporate delay guarantees, it is unlikely that the number of jobs in the system reaches this upper bound. Thus, it is safe to assume for simplicity of analysis that there is no limit on the number of simultaneous jobs. This allows us to model an application server as an unbounded Processor Sharing (PS) queueing system.

The parameters described above are assumed to be available as inputs to the provider such that it may decide the number of servers, $\eta_i$, to allocate to each customer $i$ such that $\sum_{i=1}^{m} \eta_i = n$. We assume that once a subset of servers is dedicated to a particular customer, that customer’s processing load is equitably balanced among that subset of servers. Hence, customer $i$’s requests, $1 \leq i \leq m$, arriving at rate $\Lambda_i$, are divided evenly
among its $\eta_i$ dedicated servers, such that the arrival rate to each of the $\eta_i$ servers receives requests at rate $\lambda_i = \Lambda_i / \eta_i$.

Our goal is formally stated as follows: Find the allocation $(\lambda_1, \lambda_2, \ldots, \lambda_m)$ that minimizes

$$\sum_{i=1}^{m} \sum_{k=1}^{s_i} \Lambda_i c_i P(D_i(w_{i,k}) > d_{i,k}) P(S_i = k)$$  \hfill (4.1)

such that

$$\Lambda_i / \lambda_i \geq 1, \forall 1 \leq i \leq m$$  \hfill (4.2)

$$\sum_{i=1}^{m} \Lambda_i / \lambda_i = n,$$  \hfill (4.3)

$$0 < \lambda_i E[B_i] < 1, \forall 1 \leq i \leq m$$  \hfill (4.4)

Note that because $P(D_i(w_{i,k}) > d_{i,k})$ is a function of $\lambda_i$, the objective function depends on the values of $\lambda_1, \lambda_2, \ldots, \lambda_m$. The first set of constraints in (4.2) requires that the number of servers allocated to each customer is at least one. The second constraint given by (4.3) says that the sum of allocations must be equal to the number of servers that the provider owns. The third constraint describes a condition, $\lambda_i E[B_i] < 1$, in (4.4), necessary for finite response times in the stationary regime of a single processor queueing system.

4.4 Application-tier Workload Characterization

In this section, we demonstrate that the arrival process to an e-commerce website’s application tier can be characterized over small to moderate timescales as a Poisson process. This analysis lays the foundation that allows us to model the serving system as a set of
$M/G/1/PS$ queueing systems.

### 4.4.1 Methodology

We analyze actual traces of requests for dynamic content. We apply a procedure that has been used previously by Sriram and Whitt in [99] to analyze the superposition of voice and data sources. There, they consider a random variable $V_k$ which is the sum of a consecutive sequence of $k$ interarrival times, such that $V_k = X_1 + X_2 + \ldots + X_k$ where $X_i$ is the time between the $i$th and $i + 1$st request arrivals. The index of dispersion for intervals (IDI) is defined to be

$$c_k^2 = \frac{k\sigma_k^2}{(E[V_k])^2}, \text{ where } \sigma_k^2 = E[V_k^2] - (E[V_k])^2.$$

The IDI is an estimator of inter-dependence within the arrival process. One interpretation of its values is the degree of correlation exhibited by the inter-arrival times. Another interpretation is an estimation of the level of "burstiness" that is exhibited by a process. The IDI of a Poisson process equals 1; The IDI of a renewal processes is invariant with respect to the number of samples, $k$ (for further discussion refer to [99]). In addition, the IDI analysis observes the behavior of the measured process over multiple time-scales by varying $k$, the number of consecutive samples.

The use of the IDI can easily be extended to analyze the behavior of the arrival process of HTTP requests. In this work, we use the IDI to analyze traces of a department store website. A typical HTTP trace stores a record of every HTTP transaction performed by the website that generated the trace, where each record contains arrival timestamps, the
requested URL, and the size (in bytes) of the object requested. The record, however, does not contain the time necessary (or time taken) for servicing the request. Using an HTTP trace, we partition the request records into contiguous intervals of equal duration, and the IDI values are computed within these limited-time intervals in order to increase the likelihood that the arrival process is stationary within the measurement interval. We compute the IDI for the arrival process of HTTP requests of dynamic content at the department store’s website using numerous daily traces. We also measure the rate of requests of dynamic content.

![IDI values and load over a 24-hour period (k = 20).](image)

Figure 4.1: IDI values and load over a 24-hour period (k = 20).

4.4.2 Trace Analysis

We present results of our experiments using traces from June 14th, 2001 from a department store’s e-commerce website as a representative of e-commerce services. We identify requests for dynamic content (and hence requests served by the application tier) as those that contain the character ? ("question mark") in the requested URL.
We partition the logging of requests over a 24-hour period (midnight to midnight) into intervals of 30 minutes and compute the IDI for each of the 30-minute intervals. We then construct non-overlapping sequences of $k$ consecutive inter-arrival intervals (although it is also permitted to use overlapping intervals to compute the IDI). Figure 4.1(a) plots the IDI ($y$-axis, left-hand side) and load (arrival rate) of the arrival process (along the $y$-axis, right-hand side) over a 24-hour period when $k = 20$, for a trace of requests in June 14th, 2000. The time of day in hours is varied along the $x$-axis, where 0 corresponds to 12AM (midnight). The sum of lengths of $k = 20$ consecutive inter-arrival intervals yields a time interval for computing the IDI in a range from 5 to 25 seconds. The curve labeled "load" depicts the load in requests per minute. We note that the IDI values are close to one except at times when the load is low, and that the IDI remains small even during periods where the load is rapidly increasing.

![IDI Test](image.png)

Figure 4.2: The IDI test over different timescales using traces of a dept. store website.

Figure 4.2(a) plots, using a logarithmic scale, the IDI values as $k$ is varied along the $x$-axis. The value of the IDI is close to one for values of $k$ up to 30, and then increases rapidly with further increasing $k$. We conclude that over small to moderate timescales,
the arrival process behaves like a Poisson process, but that for larger timescales, there is a high degree of correlation among arrivals. The observations presented here were also observed from traces taken on different days.

There is an intuitive explanation as to why the arrival process of requests for dynamic content presents a different behavior than the arrival process of requests of static content, whose arrivals are generally bursty. Queries for static content from a single user are often batched. The canonical example involves a web page that contains several objects, such as images and text. Hence, the interarrival times of adjacent requests to a server for static content often come from the same user and their transmission times are heavily correlated. On the other hand, the observation of low levels of correlations within the arrival process at the application tier can be explained intuitively by regarding the arrival process as a superposition of multiple users placing queries, where numerous additional queries are placed by other users between a user’s pair of queries for dynamically generated content. In contrast with an arrival process of requests of static content, a request of dynamic content (to the application serving tier) typically requires processing of a single job instead of a batch of transactions. Therefore, in timescales (seconds) on the order of lengths of interval between user actions, we expect the incoming requests to come from different users. Since each user’s actions are independent from the actions of the other users, there is little correlation in the superposition process of all arrivals within a short time period.

The intuition presented above also explains why the IDI peaks during low-load periods. During such periods, requests come from a small population of users and the
inter-arrival times of two or more requests from a same user will be highly correlated. Therefore, correlations among arrivals are expected to increase in low-load periods because of small number of sessions. For provisioning purposes, however, the behavior process during low-load periods is not of much concern, since the arrival rate \( \lambda \) used to provision servers will overestimate the arrival rate during these periods.

Figure 4.3: Characterization of workload including requests for not only dynamic content but also static content.

As a point of comparison, we also measure the IDI of traces containing all HTTP requests (not only requests directed to the application tier) and obtain remarkably different results. First, we consider the arrival process from all HTTP requests to the e-commerce server, including those for static content that are handled by the front-end serving tier. Figure 4.3 also depicts the IDI and load as in Figure 4.1(a), using the same trace of June 14th, 2000, and including not only requests for the dynamically generated content but also requests for static content. We see here that for this arrival process, the IDI values increase significantly in high load periods. In fact the IDI ramps up along the load curve. We can conclude that for the front-end serving tier, the adjacent interarrival times are highly correlated, and cannot be modeled accurately as a Poisson process.
(a) 98 Olympics (February 13th, 1998).

(b) IBM (May 22, 2001).

(c) IBM intranet (June 20, 2001).

Figure 4.4: IDI values for different traces.
We also perform a similar set of IDI tests using traces of requests for dynamically generated content from a class of websites that do not support e-commerce applications. Figures 4.4(a), 4.4(b), and 4.4(c) show measurements obtained using traces of the 98 Olympics site (trace dated from February 13th, 1998), a corporate site of IBM (trace dated from May 22, 2001) and an internal site of IBM (trace dated from June 20, 2001), respectively. Figure 4.4 plots IDI values and load, with time in hours varied along the x-axis. With the exception of the traces from the 98 Olympics site, we again find that the IDI value is small except when loads are low. For the Olympics 98 site, we find that even under high loads, the IDI value is large, indicating that there is a significant degree of correlation between interarrival times, even during short timescales. Challenger et al. in [26] argue that correlations are strong because users know the sporting events schedule beforehand. For instance, an event scheduled for 5pm will trigger user actions at approximately 5pm. Intuitively this phenomenon is expected less likely to occur for an e-commerce website since there are no “synchronized” events that would suddenly interest a large body of users.

The justification that enables us to concern ourselves with correlations only up to a certain timescale comes from the theories of three independent studies developed by Grossglauser and Bolot [51], Ryu and Elwalid [90], and Cao and Ramanan [21]. These works show that for queueing systems, the timescales over which correlations exist are delimited by an upper bound, named the Critical Time Scale in [90]. As a result, any model that accurately captures the correlation structure, including Markov models [51] up to the Critical Time Scale will closely approximate the behavior of the queueing sys-
tem. In our case, e-commerce website transactions at the application tier are expected to complete on the order of a fraction of a second. The computation of IDI values on the timescale of tens of seconds is expected to measure correlations that well exceed the Critical Time Scale.

![CDF Graph](image)

Figure 4.5: Distribution of length of sum of $k$ consecutive intervals.

In Figure 4.5, we plot the cumulative distribution function (cdf) of the time required to receive 20 arrivals, where the samples are drawn from a 30-minute interval, selected arbitrarily from one of the peak hours, whose load is approximately 100 requests per minute. The cdf values are shown along the y-axis as a function of time (seconds) which is varied along the x-axis. We see that almost always, the time required to receive 20 requests is on the order of several seconds. For instance, approximately 90% of the samples last longer than 10 seconds. Since the IDI is approximately 1 when the number of samples used per interval is 20, and since a time of several seconds intuitively lies beyond a Critical Time Scale we find that the arrival process is adequately modeled by a Poisson process.

Since the arrival traffic is effectively Poisson, an $M/G/1/PS$ queueing system is a fairly accurate model of the behavior of an application server. We utilize this model to
derive mechanisms that allocate servers to e-commerce websites (customers). In general, when a number of servers is allocated to a customer we model each server as an $M/G/1/PS$ queueing system. Assuming requests to this customer website are placed at each queueing system with equal probability, the arrival rate of requests is effectively equally split among the servers allocated to that customer. Hence, on average the same amount of work is placed upon each queueing system assigned to the same customer.

4.4.3 Analysis—why Poisson arrivals

To understand why the arrival process to the application tier behaves similarly to a Poisson process, we perform an analysis analogous to that of multiplexed voice sources in [99].

In our model, a user session consists of a sequence of user actions [67], and is modeled as a renewal process. For instance, let us consider that user clients issue transactions for actions such as “search”, “purchase”, or “view-cart”, in their user sessions, and assume initially that each of these transactions generates a batch of simultaneous requests to the application–server tier. Each renewal interval starts upon a batch arrival and is described by two additional times: the response time due to processing of the batch of requests, described by a random variable $W$, and the user think time necessary to comprehend the output of the transaction before deciding on her next action. The length of a renewal is a random variable $Y$ and its rate is given by $\lambda_r$. We consider $h$ concurrent sessions, labeled from 1 to $h$, and we assume that the renewals from a session $x$ and renewals from a session $z$ are mutually independent and described by the same renewal process distribution, $F_Y(t)$. The random variable $R_i$ is the residual time of a renewal of user
session labeled $i$.

For a single user session, the time between batch arrivals is equal to the renewal periods, thus the distribution of inter-arrival times is the distribution of renewal periods. However, during this renewal interval of a session, other sessions’ batches can arrive. We let the random variable $X$ describe the time elapsed from a batch arrival to a subsequent batch arrival (possibly from another session) when sessions are superimposed. Considering that a session $x$’s batch arrives at time $\tau_x$, if the next batch arrival at time $\tau$ comes from any of the $h - 1$ sessions other than session $x$, say session $z$, then $\tau - \tau_z$ is a residual time of the current renewal of session $z$. If the next batch arrival comes from session $x$, then the time $\tau - \tau_x$ is the renewal time $Y$. Hence, $X$ is the smallest time among $h - 1$ residual times $R$ and a renewal length $Y$. For the analysis that follows, we initially condition the inter-arrival times on the processing time $W = w$. In particular, for $0 \leq t \leq w$, $F_Y(t) = 0$, because a new batch coming from any session, say session $x$, cannot arrive before a current session $x$’s batch remains under processing. In this case, the residual time distribution (for residual times of sessions other than session $x$) is uniform in the interval $[0, w]$, i.e. $P(R \leq t) = \lambda_r \int_0^t (1 - F_Y(u))du = \lambda_r t$, $0 \leq t \leq w$. For the superposition of $h$ sessions, the distribution of time between batch arrivals is $P(X > t) = P(\min(R_1, R_2, ..., R_{x-1}, Y, R_{x+1}, ...R_h) > t) = (1 - \lambda_r t)^{h-1}$. Scaling the time by $h$ results: $P(X > t/h) = (1 - \lambda_r t/h)^{h-1}$ for $0 \leq t \leq wh$. When $h \to \infty$, $P(X > t/h) \to e^{-\lambda_r t}$, for $t > 0$. Therefore, as the number of sessions is increased, the distribution of inter-arrival times tends toward an exponential distribution. Thus, when sessions are superimposed, request arrivals are described as a batch-Poisson
process. When the average number of requests per batch is one, the time between request
arrivals, when sessions are superposed, is described by a Poisson process. Since, in prac-
tice, \( h \) is finite, we expect the Poisson description for request arrivals to be valid up to a
certain timescale, which corresponds to the notion of Critical Time Scale.

### 4.5 Server Provisioning

In this section, we derive three methods that allocate servers to customers. An exact so-
lution requires knowledge of the distribution of response times in an M/G/1/PS queue.
The solutions generated by these methods approximate the optimal solution of the opti-
mization problem posed in Section 4.3. Approximation methods are necessary for this
problem since the distribution of response times in an M/G/1/PS queue in exact form
remains an open problem (refer to a survey of known results in [118]).

#### 4.5.1 General Solution Procedure

In our original optimization problem (4.1), (4.2), (4.3), and (4.4), we replace \( P(D_i(w_{i,k}) >
d_{i,k}) \) with a bounding function, \( \psi_{i,k}(\lambda_i, w_{i,k}, d_{i,k}) \). The difference in the three methods is
the used bounding function. We seek to minimize the objective function
\[
\sum_{i=1}^{m} \sum_{k=1}^{n_i} \lambda_i c_i \psi_{i,k}(\lambda_i, w_{i,k}, d_{i,k}).
\]
Using the Karesh-Kuhn-Tucker (KKT) [23] conditions, this optimization problem can be solved via Lagrange multipliers.

First, we take the partial derivatives of the Lagrangian function for customers \( i \) and
\( j \), each with variables \( \lambda_i \) and \( \lambda_j \) respectively. For the partial derivative with respect to
variable \( \lambda_i \) (to variable \( \lambda_j \)) we are able to express the Lagrange multiplier related to the
constraint in (4.3) as a function of customer \( i \)'s (customer \( j \)'s) parameters. After equating the function of customer \( i \)'s parameters with the function of customer \( j \)'s parameters we determine \( \lambda_i \) as function of \( \lambda_j \):

\[
\sum_{k=1}^{s_i} c_i \lambda_i^2 \frac{\partial \psi_{i,k}}{\partial \lambda_i} (\lambda_i, w_{i,k}, d_{i,k}) P(S_i = k) =
\sum_{v=1}^{s_j} c_j \lambda_j^2 \frac{\partial \psi_{j,v}}{\partial \lambda_j} (\lambda_j, w_{j,v}, d_{j,v}) P(S_j(w) = v)
\]

(4.5)

We can use the above equation to solve for \( \lambda_j \) in terms of \( c_j, w_{j,v}, d_{j,v} \), \( c_i, w_{i,k}, d_{i,k} \), and \( \lambda_i \). We refer to this solution as the comparative equation. A property of the comparative equation used in the three methods presented in this Section guarantees that \( \rho_j = \lambda_j E[B_j] < 1 \). Hence, (4.4) is always satisfied. A specific \( \lambda_i \) value is obtained by applying the comparative equation for each \( j \) in terms of \( i \), excluding exceptional cases noted below, to the equality given by (4.3), resulting in a first-allocation equation.

One complication, however, is that the solution from the comparative equation may determine some set of customer's arrival rates to be greater than the actual arrival rate (for customer \( i, \lambda_i > \Lambda_i \)). Such a solution is of course not practical (unacceptable given the constraints), so a constraint from either (4.2) or 4.4 is imposed whereby \( \lambda_i = \min\{\Lambda_i, 1/E[B_i]\} \).

Thus, if the comparative equation for customer \( i \) indicates that \( \lambda_i^* \) is the solution, the final allocation is a function of \( \lambda_i = \min\{\lambda_i^*, \Lambda_i, 1/E[B_i]\} \). Since for our methods \( \lambda_i E[B_i] < 1 \), this can be rewritten as \( \lambda_i = \min\{\lambda_i^*, \Lambda_i\} \). This solution determines the first-allocation equation to be derived applying \( \lambda_i = \min\{\lambda_i^*, \Lambda_i\}, 1 \leq i \leq m \), to (4.3).

When the comparative equation solution is greater than \( \Lambda_i \), we make \( \lambda_i = \Lambda_i \), ba-
sically discarding the comparative equation solution as not valid. This basically en-
forces that the number \( \Lambda_i/\lambda_i \) of servers allocated to customer \( i \) is at least one. The
first-allocation equation, however, depends on the results of the comparative equation.
Only results that are indeed "valid", i.e. yield a result greater than or equal to \( \Lambda_i \), should
be taken into account. Hence, it is necessary to define a set \( \Phi \) to contain all customers
such that the comparative equation yields solutions greater than \( \Lambda_i \). We define \( l \) as the
cardinality of set \( \Phi \), i.e. \( l = |\Phi| \). We describe later in Section 4.5.5 how to compute fi-
nal values for \( \lambda_i \) in face of this dependency between the first-allocation and comparative
equations.

The number of servers per customer \( i \), \( \eta_i \), \( 1 \leq i \leq m \) is finally computed using a
rounding procedure applied to the ratio \( \Lambda_i/\lambda_i \) (details discussed in Section 4.5.5).

We remark that, when a server provider has a sufficient number of servers to guarantee
that \( l = m \), a single equation solves the necessary number of servers for all customers.
This equation is simply derived from the first-allocation and comparative equations, using
the fact that \( l = m \). In practice it is expected that the number \( n \) of servers is sufficiently
large (i.e., provider avoids underprovisioning its set of servers) such that \( l = m \) can also
be expected. This is of practical importance when demands for customer e-commerce
sites fluctuate. In that case, a server provider has a simple mechanism to periodically re-
allocate servers via a single equation taking into account current demand measurements.
4.5.2 Costs as function of average response times

The first method, called *Average-based method*, requires that a provider knows for every customer the average service demand times of requests placed on that customer website. These measures can be known in practice via experimentation with a customer’s application tier procedures. For a PS queue with utilization \(\rho\), \(0 < \rho < 1\), a job requiring \(w\) units of time is expected to be completed, in average, after \(w/(1 - \rho)\) units of time, due to the simultaneous processing of other jobs. The slow down factor, defined as the ratio \(D(w)/w\), is known to converge to \(1/(1 - \rho)\) for large \(w\) under any work-conserving discipline [52].

We approximate the number of service misses as a function of averages of response time. We apply the Markov inequality to find a bound to the ccdf of response time \(D(w)\) distribution

\[
P(D(w) \geq d) \leq \frac{E[D(w)]}{d} = \frac{w/d}{1 - \rho}.
\]

(4.6)

Therefore the number of service misses for a customer \(i\) at demand level \(k\) is approximated as \(\Lambda_i P(S_i = k) \frac{w_i/k/d_i/k}{1 - \rho_i}\). The optimization problem is to minimize the loss of revenue

\[
\sum_{i=1}^{m} \sum_{k=1}^{s_i} c_i \Lambda_i \frac{w_{i,k} P(S_i = k)/d_i,k}{1 - \rho_i},
\]

as described in Section 4.3, subject to the constraints given by (4.2), (4.3), and (4.4).

We derive the comparative equation \(\lambda_i = \frac{\rho_i \gamma_{i,j}/E[B_j]}{1 - \rho_j + \rho_j \gamma_{j,i}}\), where

\[
\gamma_{i,j} = \sqrt{\frac{c_i E[B_i]}{c_j E[B_j]} \frac{\sum_{k=1}^{s_i} w_{i,k} P(S_i = k)/d_i,k}{\sum_{v=1}^{s_j} w_{j,v} P(S_j = v)/d_{j,v}}}.
\]
The first-allocation equation is subsequently derived:

$$
\lambda_j = \frac{(1/E[B_j]) \sum_{k \in \Phi} \tilde{\rho}_k \gamma_{k,j}}{n - m + l - \sum_{k \in \Phi} \tilde{\rho}_k (1 - \gamma_{k,j})}
$$

(4.7)

where \( l \) is the number of customers for which the comparative equation is valid and \( \tilde{\rho}_k = \Lambda_k E[B_k] \). The constraint \( 0 < \lambda_i E[B_i] < 1, 1 \leq i \leq m \) in (4.4) is always satisfied, since, rewriting the result of the comparative equation, \( \rho_i = \frac{\rho_j \gamma_{j,i}}{1 - \rho_j + \rho_j \gamma_{j,i}} \), we find a fraction whose numerator is smaller than its denominator. Hence, the arrival rate for any \( i \) is derived to be

$$
\lambda_i = \min\left\{ \frac{\rho_j \gamma_{j,i} / E[B_i]}{1 - \rho_j + \rho_j \gamma_{j,i}}, \Lambda_i \right\}.
$$

### 4.5.3 Costs as function of variance of response times

Our second method requires knowledge of both the first and second moments of the servicing time distribution. Again, these parameters are easily estimated in practice. We call this method the *Variance-based* method, because the bound for response time distribution in this method applies Chebyshev’s inequality to bound the variance of response times. An exact expression for the variance of response times involves an integration term [117], making an exact derivation difficult. We derive another bound of the complementary distribution function using an upper bound of the variance. Consider a PS queue with average demand time \( E[B] \), second moment of general demand time \( E[B^2] \) and utilization \( \rho \). Again let an arriving request require a job demand \( w \). From the exact known result for the variance of response times in a PS queue [117], we can show that \( \text{var}[D(w)] \leq w \rho E[B^2] / (1 - \rho)^2 \). Zwart and Boxma [122] find an expression for the
second moment of the response times in an M/G/1/PS queue in which this same expression appears as an asymptotic limit when job sizes grow large. Therefore, the bound is tighter for large size jobs. We derive a second bound for the complementary distribution of response times (ccdf) in an M/G/1/PS queue. First, we apply Chebyshev’s inequality, \( P(D(w) - E[D] \geq d) \leq \frac{\text{var}[D(w)]}{d^2} \). Using the necessary condition for stability \( \rho < 1 \), we find:

\[
P(D(w) - E[D] \geq d) \leq \frac{E[B^2]}{E[B]} \frac{w}{(1 - \rho)^3 d^2}.
\]

(4.8)

### 4.5.3.1 Optimization

Using (4.8), the cost per customer is approximated using \( \psi_{i,k}(\lambda, w_{i,k}, d_{i,k}) = \frac{w_{i,k}E[B^2]}{d_{i,k}^2 (1 - \rho)^3 E[B]} \), such that the overall cost for the provider is

\[
\sum_{i=1}^{m} \sum_{k=1}^{s_i} c_i \lambda_i \frac{w_{i,k}E[B^2]}{d_{i,k}^2 (1 - \rho)^3 E[B]} P(S_i = k).
\]

The optimization via Lagrange multipliers results in allocations \( \lambda_i, 1 \leq i \leq m \) such that \( \frac{\rho_i^2 \beta_i^2}{(1 - \rho_i)^4} = \frac{\beta_i^2 \beta_i^2}{(1 - \rho_i)^4} \), where \( \beta_u = \frac{c_u E[B^2]}{(E[B_u])^2} \sum_{k=1}^{s_u} w_{u,k} P(S_u = k) / d_{u,k}^2 \). After algebraic manipulation, we obtain the comparative equation in this method:

\[
\lambda_i = \frac{1}{E[B_i]} \sqrt{1 + \frac{\beta_i \rho_i}{\beta_i (1 - \rho_i)^2}} - 1.
\]

(4.9)

The right-hand side of (4.9) is written as a product of two fractions. The second fraction is smaller than one. Furthermore, the second fraction equals \( \rho_i = \lambda_i E[B_i] \). Hence, the comparative equation for the Variance–based method yields results that satisfy
the constraint given by (4.4).

(4.3) in this case yields a radical equation containing only one customer rate to be used as a first allocation as follows. The first allocation equation for the Variance–based method is derived from (4.9) and (4.3):

\[
\sum_{i \in \Phi} \left( \Lambda_i \sqrt{1 + \frac{4 \beta_i \rho_i}{\beta_i (1 - \rho_i)^2} + 1} \right) = n - m + l
\]  

(4.10)

It is hard to isolate the first rate variable, \( \lambda_j \), to derive a first-allocation equation. Thus, a shortcoming of this method is the difficulty of finding the first allocation. Simple numerical solution procedures such as the Bisection method [42] can be applied in this case.

### 4.5.4 The EBB–based Method

Our third method utilizes bounds derived for a class of processes called Exponential Bounded Burstiness processes as defined in [116] by Yaron and Sidi, and later generalized in [101]. Zhang et al. find statistical guarantees for a generalized processor sharing discipline in [65] when the arrival process belongs to the class of EBB processes. From earlier results from [116], it is relatively simple to prove (shown in this section) that the Poisson process belongs to the class of E.B.B. processes. We can therefore use the results of Zhang et al. and establish a bound on the \textit{ccdf} of response times in our \( M/G/1/PS \) model.

The theory in [116] defines the class of E.B.B. processes to contain any process \( X(t) \) such that there exist parameters \( (\nu, \phi, \theta) \) satisfying the condition \( P(\int_s^t dX(\tau) > \nu(t-s) + \sigma) \leq \phi e^{-\theta \sigma} \). We let \( X(t) \) be a Poisson process with rate \( \lambda \), and deduce from Proposition
3 regarding a Bernoulli random process) in [116] that an \( \alpha > 0 \) can be found satisfying

\[ P(\int_s^t dX(\tau) > (\lambda + \epsilon)(t - s) + \sigma) \leq e^{-\alpha \sigma}, \]

where \( \epsilon > 0 \). Thus, a Poisson process with rate \( \lambda \) is an E.B.B. process with parameters \( (\lambda + \epsilon, 1, \alpha) \).

Since a Poisson process is time-invariant, we substitute \( N(t) = \int_s^t dX(u) \) in the previous inequality, where \( t = \tau - s \), and \( N(t) \) remains of the same nature of \( X(t) \), i.e. Poisson process with rate \( \lambda \). Here, we are able to apply a Chernoff bound, \( P(N(t) > (\lambda + \epsilon) t + \sigma) \leq e^{E[N(t)]/(\lambda + \epsilon) t + \sigma} \). By using the z-transform of the Poisson process, we find \( \alpha \) given an \( \epsilon > 0 \):

\[ P(N(t) > (\lambda + \epsilon)t + \sigma) \leq \frac{e^{-\lambda t (1-z) - \log(z) (\lambda + \epsilon) t}}{e^{\log(z) \sigma}}. \]  

(4.11)

Mapping parameters of (4.11) to the ones of \( P(\int_s^t dX(\tau) > (\lambda + \epsilon)(t - s) + \sigma) \leq e^{-\alpha \sigma} \)
yields \( P(N(t) > (\lambda + \epsilon)t + \sigma) \leq e^{-\log(z)\sigma} \), with the necessary condition \( \lambda(1 - z) + (\lambda + \epsilon)\log(z) \geq 0 \). Furthermore given \( \epsilon > 0 \), \( z > 1 \) so that the right-hand side is a decreasing function. In this case the decay parameter is given by \( \log(z) \). We pick \( z_m \), as a value for \( z \), such that the bound decays quickly. But the condition given by \( \rho (1 - z_m) + (\rho + \epsilon) \log(z_m) \geq 0 \) must be satisfied, and, increasing \( \epsilon \), we find larger values of \( z_m \) satisfying this condition. However, increasing \( \epsilon \), also relaxes the tightness of the bound, unless the desired range of response times is much farther than the mean value \( \lambda t \).

Thus, a tradeoff exists for the selection of \( \epsilon \).

For convenience, we define \( \lambda^* = \lambda + \epsilon \) and \( \rho^* = (\lambda + \epsilon) E[B] = \rho + \epsilon E[B] \). Applying the upper rate parameter, \( \lambda + \epsilon \), and the decay parameter, \( \log(z_m) \), to a result of Zhang et al. results we derive a bound on the probability of response time \( D \), (Eq. 36 of [65]):
\[ P(D \geq d) \leq \phi^* e^{-\alpha gd}, \] where \( \phi^* = \frac{\delta e^{\alpha^* \delta}}{1-e^{-\alpha(g-\rho^*)\delta}}, \) \( g \) is the fraction of processing rate for a job, and \( 0 < \delta < \frac{\log(\phi+1)}{\alpha(g-\rho^*)}. \) In our model all jobs receive equal treatment, and we use the fraction of processing rate \( g = \frac{1}{\bar{n}}, \) where \( \bar{n} \) is the number of concurrent jobs under processing. Therefore, \( P(D \geq d) \leq \phi^* e^{-\alpha gd} \) is expanded to

\[
P(D \geq d) \leq \frac{e^{\log(z_m) \rho^* \delta}}{1-e^{-\log(z_m)(g-\rho^*)\delta}} e^{-\log(z_m)gd}
\] (4.12)

We write \( \phi^* \) as a function of \( \delta. \) A suitable value for \( \delta \) is the minimum value for the function \( \phi^*(\delta). \) We select the value \( \hat{\delta} \) such that the derivative \( \phi^*(\hat{\delta}) = 0, \) \( \hat{\delta} = \frac{\log(\phi^*) - \log(\rho^*)}{\alpha(g-\rho^*)}. \)

By applying this result to (4.12), we find a third bound for the \( ccdf \) of response times to be applied in our framework:

\[
P(D > d) \leq \frac{\gamma^{\gamma-1}}{1-\gamma} e^{-\log(z_m) \frac{1-\delta}{\rho^*} d} \] (4.13)

where \( \gamma = \rho^*/g. \)

### 4.5.4.1 Optimization

We derive here the comparative and first-allocation equations for the method based on the EBB model. A drawback of the EBB method is that it only permits a single mapping of \( d_i \) to \( w_i, \) but such a condition is often used in practice. A natural value of \( w_i \) is \( E[B_i], \) as discussed in Section 4.3. Here we further bound the \( ccdf \) of the response times to a function \( \psi_i(\lambda_i, w_i, d_i) \) that yields a more conservative cost function whose solution is tractable. In order to find \( \psi_i(\lambda_i, w_i, d_i) \) we derive the inequality departing from the bound
in (4.13)

\[
\frac{\gamma^{2/3}}{1 - \gamma} e^{\log(\varepsilon_m) \frac{1}{\rho_i} \log d_i} < \frac{e}{1 - \gamma} e^{\log(\varepsilon_m) \frac{1}{\rho_i} \log d_i} < (1 + \epsilon') e^{1 - \log(\varepsilon_m) \frac{1}{\rho_i} \log d_i} \tag{4.14}
\]

which results in \( \psi_i(\lambda_i, w_i, d_i) \) defined for the EBB-based method as in the last term of the above inequality. Thus, \( \psi_i(\lambda_i, w_i, d_i) = (1 + \epsilon') e^{1 - \log(\varepsilon_m) \frac{1}{\rho_i} \log d_i} \). The inequality is valid only if \( \rho_i < \hat{\rho}_i \), where \( \hat{\rho}_i \) is any \( \rho_i \) such that \( 1/(1 - \gamma) < 1 + \epsilon' \), \( \epsilon' > 0 \). Thus, here we add an extra constraint to the original problem.

We solve the problem of allocation, as defined in our framework, minimizing the total cost function \( \sum_{i=1}^{m} c_i \Lambda_i \psi_i(\lambda_i, w_i, d_i) P(S_i = k) \). Since the constant \( \epsilon' \) is used across all customers as a multiplicative constant, we need to solve the optimization problem as formulated in our general model (Section 4.3) with the function \( \psi_i(\lambda_i, w_i, d_i) = c_i \Lambda_i e^{1 - \log(\varepsilon_m) \frac{1}{\rho_i} \log d_i} \).

We find the comparative equation for pairs of variables \( \lambda_i \) and \( \lambda_j \), \( 1 \leq i < j \leq 1 \),

\[
\lambda_j = \lambda_i \frac{d_j E[B_i]/E[B_j]}{d_i + \lambda_i E[B_i] \left( \frac{\log(\frac{c_i d_i E[B_i]}{c_j d_j E[B_j]})}{\log(\varepsilon_m)} + d_j - d_i \right)}.
\]

The first-allocation equation is obtained using (4.3) for \( \lambda_i \):

\[
\lambda_i = \frac{(d_i / E[B_i]) \sum_{j \in \Phi} \Lambda_j E[B_j]/d_j}{n + l - m - \sum_{j \in \Phi} \frac{\Lambda_j E[B_j]}{d_j} \left( \frac{\log(\frac{c_i d_i E[B_i]}{c_j d_j E[B_j]})}{\log(\varepsilon_m)} + d_j - d_i \right)}.
\]
4.5.5 Solving allocations via comparative and first-allocation equations

The typical procedure to find allocations requires that one first constructs the first-allocation equation and determines the value of one of the rates, say $\lambda_i$. This equation makes use of a parameter for $l$, which is the number of customers for which the comparative equation is ultimately valid. However, $l$ can only be identified once the values for $\lambda_j$, $1 \leq j \leq m$ are known. Here, we present an algorithm that iterates over candidate values of $\lambda_j$, $1 \leq j \leq m$, and $l$ until an appropriate solution is found.

Let us assume that a first allocation is computed using a value of $l$ equal to $\hat{l}$, yet to be found. If after all comparative equations are computed and the resulting $l$ equals $\hat{l}$, then the first allocation equation is computed such that the general conditions of the problem are satisfied. Therefore, the procedure to select values for $\lambda_i$ must select a value for $\hat{l}$.

A natural first choice is to set $l = m$ and $\Phi = \{1, 2, ..., m\}$ for the first-allocation equation. Using the $\lambda_j$ value found via the first-allocation equation, each $\lambda_i$ is computed since the comparative equation gives $\lambda_i$ in terms of customer $i$'s and $j$'s parameters. The algorithm keeps a customer $i$ in $\Phi$, if $\Lambda_i/\lambda_i \geq 1$. If $\Lambda_i/\lambda_i < 1$, the customer is placed out of $\Phi$ and its $\lambda_i$ is set to $\Lambda_i$. The value of $l$ is re-evaluated by the cardinality of $\Phi$. If $l$ is re-evaluated equal to its previous value, the allocations satisfy the constraints of the problem. In this case the algorithm stops and outputs the set $(\lambda_1, ..., \lambda_m)$. Otherwise, the sequence of computations of first-allocation equation and comparative equations is repeated with the new value of $l$ and the re-arranged set $\Phi$. In the worst-case, the sequence of first-allocation plus comparative equations is repeated at most $m$ times.

The selection of the index $j$ for a $\lambda_j$ value to be obtained in computations of first-
allocation equation does not affect the result, if an extra procedure is taken, since from
the theory $\lambda_j$ is a solution of a set of $l$ equations and $l$ variables. The extra procedure
involves assuring that the index $j$ is such that the comparative equation is valid for $\lambda_j$
in the final allocation. Hence an index $j$ must be selected for a first allocation such that
$\Lambda_j/\lambda_j \geq 1$ is likely. An intuitive idea to find such index is to choose the index whose
partial cost $\sum_{k=1}^{s_i} c_i \Lambda_i \psi_{i,k}(\lambda_i, w_{i,k}, d_{i,k}) P(S_i = k)$ is maximum among partial costs of all
customers, since a high partial cost implies high likelihood that $\Lambda_j/\lambda_j \geq 1$.

The number of servers for a customer $i$ is determined from the value $\Lambda_i/\lambda_i$. This ratio
does not necessarily result an integer value, hence a rounding procedure is necessary. We
find the number $y = \sum_{i=1}^{m} \Lambda_i/\lambda_i - \sum_{i=1}^{m} [\Lambda_i/\lambda_i]$, where $[x]$ denotes the maximum inte-
ger smaller than $x$, and to construct the set $\Psi(y)$ containing the $y$ customers whose fractional portions of their costs are largest when estimated via $\sum_{k=1}^{s_i} c_i \Lambda_i \psi_{i,k}(\lambda_i, w_{i,k}, d_{i,k})$.
Finally, we determine the number of servers $\eta_i$ to be $\eta_i = [\Lambda_i/\lambda_i] + 1$, if $i \in \Psi(y)$, and
$[\Lambda_i/\lambda_i]$, otherwise.

4.6 Experiments

We perform a series of experiments via discrete–event simulation to evaluate costs for
a variety of partitioning configurations of a provider's servers. For our simulations we
utilize the model with Poisson arrivals and processor sharing queues in order to describe
application servers. We resort to synthetic workloads for a number of customers, since we
had traces of only one e-commerce website, and no knowledge of processing times for the
requests contained in the traces. By using synthetic workloads we have the ability to fully
characterize customers as a function of their sensitivity to response time and intensity of the workload. We compare the costs that result from application of the Average, Variance and EBB-based methods developed in the previous section to the costs obtained by simple heuristic approaches. A first heuristic sets the number of servers in proportion to a customer arrival rate times the charge per request, divided by the tolerated response time. We name this heuristic “naive”. A second heuristic, termed “uniform”, divides the servers evenly among existing customers. The costs produced by these methods and heuristics is compared to an estimate of the minimum cost obtained via Monte-Carlo simulation. For each experiment, an iteration of the Monte-Carlo simulation assigns servers at random among the customers that assures that each customer is assigned at least one server. We perform one million iterations per experiment, and return the lowest cost obtained.

To our knowledge, there is no study that parameterizes the time necessary to execute applications in an operating system of an application server. However, the standard SPECWeb99 used to evaluate Web server performance uses a lognormal distribution for file sizes [76]. We select a lognormal distribution to describe the distributions of $B_i$, $1 \leq i \leq m$, for this reason, and also because it permits one to vary both first and second moments.

We compare the various methods and heuristics over a suite of specific configurations of customers. Each customer $i$, $1 \leq i \leq m$ can either be tolerant (labeled 'T'), when the SLA specifies a required response time for a query of less than $8E[B_i]$. A customer's SLA is severe (labeled 'S') when its required response time is $2E[B_i]$. In addition, each customer's intensity, $\bar{\rho}_i \equiv \Lambda_i E[B_i]$ can be high (labeled 'H') such that we set $\bar{\rho}_i = n/m$,
or can be low (labeled 'L') such that we set $\tilde{p}_i = 0.2n/m$. We set the standard deviation of servicing times equal to the average size $E[B_i]$, thus the variance is $(E[B_i])^2$. Hence, a customer $i$ belongs to one of four classes: TL, TH, SL, or SH. We describe a configuration as a vector containing customer description labels: $(e_1, \ldots, e_m)$, where $e_i \in \{\text{TL, TH, SL, SH}\}, 1 \leq i \leq m$. We assume only one response time level per customer, i.e. $\forall i, s_i = 1$. In this case, the demand $w_i = E[B_i]$. We assume the charges $c_i, 1 \leq i \leq m$, are 1 for simplicity. In the absence of a first-rate allocation equation for the variance method, we solve the root for the radical equation via the Bisection method [42].

### 4.6.1 Experimental Results

In our first set of experiments, we compare the costs yielded by the various methods and heuristics for five different customer configurations for a provider supporting $m = 4$ customers with $n = 20$ servers. The configurations are: A = (SH, SL, SL, SL); B = (SH, SH, SL, SL); C = (SH, TL, TH, SL); D = (SH, TH, SL, SL); E = (SL, SL, SL, SL). Figure 4.6 plots along the y-axis the costs incurred for the various allocations under these five different customer configurations. The bars labeled "average", "variance", "EBB", "uniform", "naive", and "MC" (Monte-Carlo) indicate costs incurred for the server allocation selected by the respective method (average, variance, and EBB), heuristic (uniform and naive), or Monte-Carlo estimated minimum (MC).

The results demonstrate that our methods achieve allocations whose costs are close to the minimum possible cost (with high likelihood), while the simple heuristic approaches generally incur significantly higher costs. In configurations C and D, for instance, the
Figure 4.6: Different configurations under different provisioning schemes.

(a) Configuration (Sx, SL, SL, TL) as base. (b) Configuration (Sx, TL, TH, SL) as base.

Figure 4.7: Costs incurred when varying only one customer (1) arrival rate ($\Lambda_1$).
cost obtained via either of the three methods is approximately 40% of the cost obtained when using an uniform allocation. Configuration E, however, is a singular case, where the four customers are identical. It is not surprising that in this case, the heuristics also work well. In general, the variance method comes closest to the minimum cost, with the average method and the EBB method usually yielding a slightly higher cost.

In our second set of experiments, we select two base-configurations and vary the arrival rate of a single customer in a system with $n = 20$ servers. Results are shown in Figure 4.7. The overall arrival rate for the first customer, $\Lambda_1$, is varied along the $x$-axis. The cost values appear along the $y$-axis. The configuration labeled “Sx” denotes that the overall arrival rate of the associated customer varies along the $x$-axis and is not simply classified as high or low. The curve labeled “Monte-Carlo” depicts the minimum cost estimated via Monte-Carlo simulation. The curves labeled “Average”, “Variance”, “EBB” depict results for the Average-based, Variance-based, EBB-based methods, respectively, and the curves labeled “uniform” and “naive” depict the cost for the heuristics’ methods.

In Figure 4.7(a) the base-configuration is (Sx, SL, SL, TL). We find that the Average-based method and the EBB-based method yield configurations whose costs are almost identical to those achieved from the configuration discovered by the Monte-Carlo simulation, and the variance method’s resulting cost relatively close to the cost obtained via Monte-Carlo simulation. The most important result, however, is that cost under all three methods increases slowly with additional load (increasing arrival rate), whereas for the uniform and naive heuristics, costs increase rapidly with higher loads.

In Figure 4.7(b), the base-configuration is (Sx, TL, TH, SL). Figure 4.7(b) demon-
strates the increase in cost that a provider will incur if the servers are allocated to customers using simple heuristics. The costs that result from applying the Average-based, Variance-based, and EBB-based methods are close to the estimated minimum cost.

![Graph showing costs](image)

Figure 4.8: Costs obtained when varying the number of customers.

In Figure 4.8, we analyze the cost generated by configurations selected via the methods and heuristics as the number of customers is varied along the $x$-axis. The costs are shown along the $y$-axis. Here, the provider offers 60 servers, where, in each experiment, a customer’s configuration is set at random to TL, TH, SL, and SH with respective probabilities 0.6, 0.15, 0.15, and 0.1. We see that, irrespective of the number of customers, the costs achieved by the method-produced configurations are near-optimal, whereas the costs achieved by the heuristic-produced configurations are significantly larger.
4.7 Determining an optimal number of servers to deploy

Previously we assumed \( n \) was a given parameter of the problem. Here, we show how to use the comparative equation and first allocation equations, in both Average–based and EBB methods, to find the optimal number of servers where there is a cost \( p \) for each server used by the provider. We apply rates \( (\lambda_1^*, \ldots, \lambda_m^*) \), obtained by the comparative and first–allocation equations to the cost function, to define the function \( \Omega(n) = \sum_{i=1}^{m} \sum_{k=1}^{s_i} c_i \psi_{i,k}(\lambda_i^*, \omega_{i,k}, d_{i,k}) \).

We assume that the SLAs are contracted such that all rates \( (\lambda_1^*, \ldots, \lambda_m^*) \) are given by the comparative equation, thus \( l = m \). In practical circumstances, both a provider and a customer should be expected to agree on an SLA for which this condition holds true. The provider’s cost \( \Omega(n) \) is still expressed as a sum of partial costs, more exactly in the form \( \Omega(n) = \sum_{i=1}^{m} a_i \frac{n + g_i}{n + g_i - f_i} \), where \( a_i, g_i, \) and \( f_i \) are constants that depend on the adopted method. Assuming that instantiation of servers cost a provider \( p \) units per server, a provider can estimate the necessary number of servers to minimize the cost by matching the derivative of the cost function with the price \( p \) such that

\[
p - \sum_{i=1}^{m} a_i \frac{f_i}{(n + g_i - f_i)^2} = 0. \quad (4.15)
\]
For instance, for the Average–based method, the constants $a_i$, $g_i$ and $f_i$ are:

\[
a_i = c_i \Lambda_i \sum_{k=1}^{s_i} \frac{w_{i,k}}{d_{i,k}} P(S_i = k),
\]

\[
g_i = \gamma_{j,i} \sum_{v=1}^{m} \hat{\rho}_v (\gamma_{v,j} - 1),
\]

\[
f_i = \gamma_{j,i} \sum_{v=1}^{m} \hat{\rho}_v \gamma_{v,j},
\]

where $j$ is taken arbitrarily, $1 \leq j \leq m$. The roots for $(4.15)$ can be found via numerical methods. The procedure of finding the adequate number of servers is especially useful for a provider that needs to make adjustments “on demand”.

### 4.8 Conclusion

We study methods to provision an e-commerce service provider’s application-tier servers among a set of customer companies to maximize the provider’s profit. We use as our setting the standard business model in which the provider forms a separate service level agreements (SLA) with each customer that indicates the profit per transaction, bound on delivery time, and the charge penalty for failing to meet the delivery time bound. We devised three methods for allocating a fixed number of servers among an arbitrary set of customers with a variety of traffic demands and different SLA configurations. The first method uses only an estimation of average response time to evaluate costs for the provider. The second method utilizes an estimation of variance of response times to evaluate costs for the provider. The third method utilizes a Poisson process description under the E.B.B.
model.

Analysis of traces revealed that it is reasonable to treat the arrival process of requests from a customer to the provider's application tier as a Poisson process. This allowed us to setup an optimization problem in the context of a set of $M/G/1/PS$ queueing systems.

Via Monte-carlo simulation, we showed that the costs of the derived methods are near-optimal, and are significantly lower than more naive heuristic methods. We conclude that application of these methods by a provider to provision its application tier serving resources offers a low complexity solution that can significantly increase profits.
Chapter 5

Summary and Future Work

5.1 Summary of the Dissertation

In this dissertation, we have proposed models and algorithms for the resource management of large-scale online services. Several services exhibit large demands, and these require from their providers an infrastructure able to serve such a demand. A centralized solution, such as having a single server, is simple to implement but presents drawbacks since it is prone to failure and does not scale. An evolved servicing architecture segments occurrences of the service into subsets such that occurrences in each of these subsets are served by a single server. Such a solution scales better but is not adaptable to the fluctuation of service demands and might result in greater response times. It is important to overcome these drawbacks, since fluctuations in demand caused by variations of popularity of servicing instances occur in many services [71, 5]. Such services include e-commerce, online auctions, and content delivery. Even if this static provisioning solution is load balanced for the demand at a particular point in time, such fluctuations in demand may cause servers to be so heavily loaded that either service is denied or processing of
the service is slow at heavily loaded servers.

Applying the concepts of load-sharing and dynamic re-provisioning can result in services that are often more scalable than those of the static provisioning solution, and that are faster and more resilient. The three scenarios that we describe in this thesis to apply these concepts are as follows.

- In services such as content distribution (e.g. distribution of a news video or Internet-worm recovery program), surges in demand can occur, and servers hosting the content and serving it become heavily loaded. Requests are denied in order to avoid overloads; but this is an unsatisfactory solution. For the provider of these services, profit is a function of the portion of service that is effectively provided. Hence, denying service reduces the providers profit. As a result, such providers must make sure that service rejection is reduced. Multiple providers can pool together their servers in order to reduce the service rejection.

- In services such as online auctions, a provider can distribute service occurrences for the same auction, e.g. bid placements, at multiple servers instead of at a single server. Since demands can fluctuate, e.g. an auction might become popular, occurrences originally going to a heavily loaded server can be distributed to multiple servers so that load is reduced at the heavily loaded server and not “surpassed” by loads at the other “sharing” servers. As a result, response times are decreased.

- A third-party server provider can provide servers that both host the content and serve it on behalf of service providers under a service level agreement. Such an idea relies on principles of economies of scale, such that this server provider can
make a profit and the service providers have the performance guarantees that they need without a need to worry about the specifics of the resource management. This can be a win-win arrangement for service provider and server provider.

There are, however, interesting questions that still must be addressed. In the case of service rejection, heterogeneous demands cause providers to overuse resources from other providers. As a result, these providers might retreat from their participation in such an agreement. The goal, however, is to prevent service rejection when demand fluctuates. We describe the concept of pooling together servers with the abstraction of a collective, modeled as a queueing system. In particular, for services such as video-streaming characterized by fixed-rate transfers, we describe a delivery system by an $M/G/k/k$ queue, where $k$ is the number of servicing units and the maximum number of transfers that can be admitted. For elastic transfers, the model for a collective of servers is an $M/G/1/k/PS$ queue, where $k$ is the maximum number of ongoing transfers in process. We define a homogeneous system, whose servers have equal demands and equal servicing units for all providers. By comparing rejection rates obtained when providers choose to form collectives of their servers to the rejection rates achieved when providers choose to operate in an isolated manner, we reach important conclusions.

We highlight three main contributions here. First, we find that for slight deviations from a system with homogeneous demands, providers quickly become unwilling to participate in a collective. Second, we study thresholding mechanisms that can extend the heterogeneity that all providers accept as a win-only solution. The three thresholding mechanisms proposed here offer protections against overuse of resources. Third, we also
build a prototype of collective management for video–streaming. Experiments performed in the PlanetLab demonstrate the concept and the usefulness of the thresholding mechanisms.

For services such as online auctions, we identified a fundamental problem that when the service is simultaneously hosted on various servers, the data for such services must be kept consistent across those servers. We find the concept of load sharing to be useful, but there is a tradeoff for the sharing factor in that servers are kept lightly loaded. Consistency imposes a cost, and it is important to estimate the impact of load sharing in these kinds of services. We have studied algorithms for resource management in the case in which fluctuation of demands occur at large time scales. This is the case of diurnal variations exhibited in a number of services correlated to people routine tasks, such as peaks on evenings. We show that greedy algorithms can significantly reduce the loads at heavily loaded servers. The fact that simple heuristics, such as greedy ones, perform well is important because the problem is NP–hard. We also extend our model to the case in which fast fluctuation of demands occur, i.e., demands fluctuate at small time-scales. Here the interesting question relates to the situation in which loads are balanced on average. The intuition is that no load sharing should be used, when compared to a static provisioning solution, because it would introduce costs due to consistency. We observe, however, that by using the fact that some demands are high while others are low at certain periods, there are multiplexing gains that can be obtained.

We highlight the three most important contributions. First, we analyze greedy algorithms and provide bounds for the worst-case output of these algorithms. Second, we
show representative cases in which the heavily loaded servers have their loads reduced to close to the average intensity of servers, which indicates that greedy algorithms are good in these cases. Third, we demonstrate that when servers are balanced in average but with demands occurring at high rates, load sharing can provide an extra gain, i.e., it reduces the average response times at servers.

Finally, for the business of providing servers, we are interested in simple models that accurately describe the profits made such that the service level agreement with service providers is maintained. The goal is to maximize profits. A first contribution here is that we find through characterization of demands that an accurate model for the application server tier of an e-commerce website is an $M/G/1/PS$ queueing system. We pose an optimization framework for the question of profits related to the response time given in such queueing systems. Since a closed-form equation for the response time of a processor sharing queue is still an open problem, we study bounds on the distribution of the response times. This permits us to formulate three methods for allocating servers to distinct service providers (e-commerce websites) while maintaining their service level agreements. We find through simulation that these three methods output results that are near-optimal and better than the results given by simple heuristics. The main contribution is that since all of these methods are simple, a dynamic re-provisioning solution can be easily performed periodically with near-optimal profits for the server provider.
5.2 Future Research Directions

There are several possibilities for future research extensions on topics investigated in this thesis. In this section, we describe some specific directions to advance the work done in this thesis and more open-ended possibilities as well.

A possible way to extend the work on collectives described in Chapter 2 is to have a single model for mixed elastic transfers and fixed-rate transfers. Since we compare systems in which either elastic transfers or fixed-rate transfers exist, it is worth investigating if the case of "mixed" transfers yields the same conclusions. The work in [19] provides results on the throughput achieved when fixed rate transfers and elastic transfers co-exist. Hence, such a model can be leveraged as a first lead to a model that describes collectives serving both elastic and fixed-rate transfers.

Another specific matter for investigation is preemptive thresholding mechanisms. The mechanisms that we study here are non-preemptive since we assume that once providers are bound by the collective arrangement, they honor requests and an ongoing transfer cannot be dropped once it has already started. By contrast, if transfers can be interrupted, such events should be counted as part of the rejection rates since providers do not profit from the interrupted transfers. In the case of fixed-rate transfers, the quality of the service is sensitive to any kind of interruption and it is not worth resuming a session since the quality of the service has been already severely affected. In the case of elastic transfers, however, it is possible to accept a momentary interruption of service if the transfer is subsequently resumed. This, of course, requires the model to be extended to accept a maximum time of interruption, making the analysis of these systems more complex.
In the case of service with consistency requirements, it would be of interest to construct a more detailed model for the fast fluctuation of demands. We focused on a demand described by two modulated levels, but models that describe demands at multiple levels or continuous levels are interesting for practical purposes. More elaborate models are more complex to analyze, though.

A more open-ended front of research is in the area of grid computing services. We basically study, in this thesis, large demands due to a large volume of requests. It is possible in grid services that a low volume of tasks are performed but that each task requires a huge amount of computation, e.g. tasks to be performed in protein folding simulations as described in [96]. In this case computer users may donate computing resources up to a defined threshold to be used for these intense computational tasks when their machines are not being locally used. If these tasks are parallelizable to some degree, then it may be worthwhile to take advantage of idle servers, if any exist in the server infrastructure. It would be interesting to know when it is of advantage to parallelize, given the costs of placing a subtasks on various machines and collecting results and also taking into account users' thresholds.

Finally, there are companies currently investing in hardware implementations of content-based information retrieval architectures for services such as search engines. The architecture uses a single chip as a building block that can scale by cascading multiple chips. In this case, power consumption is one the most important metrics of interest. Such a metric poses a different goal from the ones that we address in this work. It seems

---

5.1 information known from private conversation
promising to study load-sharing with the goal of reducing power consumption in these kinds of serving architectures.
Bibliography


[53] HENNING, M. Massively multiplayer middleware. *ACM Queue* 1, 10 (Feb. 2004), 38–45.


Appendix A

Rejection Rate in Processor Sharing Model

In order to find the distribution of number of transfers under the elastic transfer model, we follow a procedure similar to the derivation described in [89] (pp. 171) for an unbounded system. The work introduced to the system by an accepted session is a random variable $U$. We denote its distribution $U(x)$ and the density probability function $u(x)$. Furthermore, we make use of the function $\bar{U}(.)$ defined as $\bar{U}(x) = P(U > x)$. The hazard function is defined as the ratio $w(x) = u(x)/\bar{U}(x)$. Work is processed at rate 1.

We define states of the system as the current completed fraction of service relative to the individual jobs in increasing order. For example, given $n$ jobs we have the state $x = (x_1, x_2, x_3, \ldots x_m)$. Since each job receives $1/m$-th of the processing, a transition from state $x = (x_1, x_2, \ldots x_i, \ldots x_m)$ to state $e_i(x) = (x_1, x_2, \ldots x_{i-1}, x_{i+1}, \ldots x_m)$ occurs with probability intensity ("rate") $w(x_i)/m$. On the other hand for the reverse process the transition from $e_i(x)$ to the state $x$ is given by $\lambda u(x_i)$. In fact we conjecture that we may construct a reverse process with balancing equations given by this transition rates and use
the theorem shown in [89] (pp. 164). We shall have

\[ p(x) \frac{w(x_i)}{m} = p(e_i(x)) \lambda u(x_i) \]  
(A.1)

We assume without loss of generality that \( x_1 \leq x_2 \leq x_3 \ldots \leq x_m \). From the definition of the hazard function we rewrite Equation A.1 as

\[ p(x) = n p(e_i(x)) \lambda \bar{U}(x_i) \]  
(A.2)

Using the definition of \( w(.) \) we are able to find the probability of states as recursively as the number in the system decreases:

\[ P(x) = P(x = (x_1, x_2, \ldots, x_m)) = P(N = 0) n! \lambda^m \prod_{i=1}^{m} \bar{U}(x_i). \]  
(A.3)

We find the expression for the number of sessions in the system as:

\[ P(N = m) = m! \lambda^m P(N = 0) \int_{x_1} \int_{x_2} \ldots \int_{x_m} \bar{B}(x_i) dx_1 dx_2 \ldots dx_m \]  
(A.4)

\[ = \lambda^m P(N = 0) \int \int \ldots \int_{x_1 \leq x_2 \leq \ldots \leq x_m} \bar{U}(x_i) dx_1 dx_2 \ldots dx_m \]

\[ = P(N = 0)(\lambda E[U])^m. \]

We can easily find \( P(N = 0) \) by applying the normalizing condition \( \sum_{m=1}^{k} P(N = m) = 1 \), thus finding:

\[ P(N = 0) = \frac{1 - \lambda E[U]}{1 - (\lambda E[U])^{k+1}} \]  
(A.5)
Hence, the blocking probability is the probability of the event of maximum utilization which is given by

\[ p_1 = (\lambda E[U])^k \frac{1 - \lambda E[U]}{1 - (\lambda E[U])^{k+1}}. \]  \hspace{1cm} (A.6)

Since we verify that \( p(x) = p(0, x)/(n + 1) \), we validate the initial conjecture.
Appendix B

Outline on the implementation of the protocol of SERES platform

Among the tasks necessary for instantiating a server collective, we include for instance the adherence of new servers to the collective, report messages on where to find content replicas, and inquiries during an overloaded state. All of these tasks involve interactions among SERES participants requiring a common "understanding" within a collective. The SERES protocol is defined to coordinate intra-collective actions. As an additional signaling the protocol should be lightweight, while still capable of all necessary actions that take place intra-collective. We define thus a small but exhaustive number of primitives that enable interaction among participants of a collective. Members of the collective use the same channel, e.g. TCP connections, for exchanging protocol messages. Underlying and parallel signaling (RTSP, RTCP) and data transport (RTP) protocols are maintained independent from the SERES protocol.
B.1 Adherence to SERES

A JOIN primitive allows a server to adhere to a collective. Providers willing to take part in a collective configure their servers to contact one of the existing members of the collective with a JOIN primitive. The JOIN primitive carries the server identification (DNS name) and specifies the amount of resources the server is willing to contribute to the collective. A collective expects a collective-global minimum resource contribution from each participant in order to avoid a “free rider” problem. The contacted SERES participant examines if the contribution proposal from a new server meets the global minimum resource contribution. Typically a JOIN primitive is a two-way transaction. The positive JOIN response brings a list of all SERES participants in the collective. The recently-joined participant becomes able to inform other participants about its content.

When leaving a collective, a SERES participant must inform one of the SERES participants about its decision. The LEAVE message is used in this context. As a “follow-up” procedure, the contacted SERES participant becomes responsible to refresh other participants via UPDATE messages.

B.2 Content Exchange

SERES servers inform one another about the location from which their content can be replicated from via LOCATOR messages. The LOCATOR primitive carries a Uniform Resource Locator (URL) indicating the location of their content. The protocol for replicating content is not of scope of SERES, for it is an orthogonal issue. There is though a
requirement of reliability for the content copies needs to be exact replicas. Protocols such as HTTP and FTP are suitable options. The LOCATOR response is sent after the content has been retrieved or after a timeout. If the content is successfully replicated it should contain the URL for eventual redirections at the newly-mirrored location, i.e. a streaming media URL.

B.3 Redirection Inquiries

New session requests that trigger inquiries for assistance from a SERES participant to other servers in the collective proceed via YOUFREE primitive. Whenever a request is to be rejected by the streaming media server, a local SERES instance can inquiry other SERES instances in the collective listed in its SERES table issuing YOUFREE messages that include the presentation identification given by the original URL in the server that hosts it as primary content. The YOUFREE responses determine to the SERES inquirer about an eventual confirmation to forward the clients' requests to the agreed SERES participant. Currently, a SERES participant inquires in sequence other members of the collective via unicast YOUFREE messages on a per-request basis. When receiving a negative YOUFREE response, a provider continues with YOUFREE attempts while there are unattempted alternatives listed in its SERES table. Typically, the YOUFREE primitive specifies the presentation identification given by its URL in the original server. The YOUFREE response indicates the result of the inquiry with an eventual error code.

Positive YOUFREE responses determine SERES to inform the streaming service the URL for redirection. Subsequently the streaming service is able to notify the client via
one of its redirection messages, e.g. a 302 Moved Temporarily for RTSP. Any subsequent streaming intra-session request, e.g., an RTSP PLAY message, is sent directly to the appointed server. SERES thus does not require any modification on client-side software. This fact allows incremental deployment with legacy systems. In case of redirections an extra latency is required in the session opening phase only. We believe that the additional latency can be generally considered minor for streaming services where sessions usually last for a relative long time.

B.4 SERES Table

SERES participants need to store limited data about the SERES collective in the SERES table. Each host is responsible for maintaining its own SERES table with data relevant to its eventual SERES actions of interest. Among the necessary data to be stored we include the original URL of primary content, a local Redirection URL, location where the content can be replicated, local URLs of non-primary content, and general collective-related usage data.

B.5 Examples

To illustrate a joining action followed by exchange of content, we show in Figure B.1 how a provider $S_2$ joins a collective contacting initially a SERES participant, server $S_1$.\textsuperscript{B.1} To start, $S_2$ sends a JOIN message to $S_1$. A JOIN OK response is returned, and $S_2$ is then

\textsuperscript{B.1}Following a "tradition" of protocols such as HTTP and RTSP, primitives are sent through ASCII messages.
\[ S_2 \rightarrow S_1 \text{ JOIN s2.org} \]
\[ S_1 \rightarrow S_2 \text{ JOIN 200 OK} \]
\[ \text{Content-List: [rtsp://s1.org/foo.mov]} \]
\[ S_2 \rightarrow S_1 \text{ LOCATOR rtsp://s2.org/bar.mov} \]
\[ \text{Content-Location: http://w.s2/bar.mov} \]
\[ \text{... (fetch http://w.s2/bar.mov) ...} \]
\[ S_1 \rightarrow S_2 \text{ LOCATOR 200 OK} \]
\[ \text{Location: rtsp://s1.org/bar.mov} \]
\[ S_1 \rightarrow S_2 \text{ LOCATOR rtsp://s1.org/foo.mov} \]
\[ \text{Content-Location: http://w.s1/foo.mov} \]
\[ \text{... (fetch http://w.s1/foo.mov) ...} \]
\[ S_2 \rightarrow S_1 \text{ LOCATOR 200 OK} \]
\[ \text{Location: rtsp://s2.org/foo.mov} \]

Figure B.1: Server \( S_2 \)'s adherence to a collective.

\[ C \rightarrow S_1 \text{ SETUP rtsp://s1.org/foo.mov RTSP/1.0} \]
\[ S_1 \rightarrow S_2 \text{ YOUFREE rtsp://s1.org/foo.mov} \]
\[ S_2 \rightarrow S_1 \text{ YOUFREE 200 OK} \]
\[ S_1 \rightarrow C \text{ RTSP/1.0 302 Moved Temporarily} \]
\[ \text{Location: rtsp://s2.org/foo.mov} \]

Figure B.2: Server \( S_1 \) inquiry \( S_2 \) to deliver a session for client \( C \).

able to issue a LOCATOR message to each of the participants read from the Content-list returned in the JOIN response. In this example \( S_2 \) proceeds informing \( S_1 \) about its object location. The server \( S_1 \) after receiving a LOCATOR message from \( S_2 \), fetches \( S_2 \)'s content to replicate it locally and responds with a LOCATOR OK which specifies the available URL for redirection. The same sequence of operations takes place inverting positions when \( S_1 \) also informs \( S_2 \) about its primary content, sending a LOCATOR message to \( S_2 \).

Figure B.2 shows an example of a sequence of messages for a typical redirection.

First, an RTSP request comes from a client \( C \) to server \( S_1 \). Considering that \( S_1 \) cannot service its content to \( C \) for it does not meet its MLRU at the moment, it contacts server \( S_2 \) with a YOUFREE message. When \( S_2 \) responds with a YOUFREE 200 OK message, then \( S_1 \) ready to transmit to client \( C \) a 302 Moved Temporarily response with the
redirection URL taken from the YOUPREE response.