University of Newcastle upon Tyne

School of Computing Science

Resource Allocation Policies for Service Provisioning Systems

by

Jennie Palmer

PhD Thesis

2006

Abstract

This thesis is concerned with maximising the efficiency of *hosting* of *service provisioning* systems consisting of clusters or networks of servers. The tools employed are those of probabilistic modelling, optimization and simulation. First, a system where the servers in a cluster may be switched dynamically and preemptively from one kind of work to another is examined. The demand consists of two job types joining separate queues, with different arrival and service characteristics, and also different relative importance represented by appropriate holding costs. The switching of a server from queue i to queue j incurs a cost which may be monetary or may involve a period of unavailability. The optimal switching policy is obtained numerically by solving a dynamic programming equation. Two heuristic policies – one static and one dynamic – are evaluated by simulation and are compared to the optimal policy. The dynamic heuristic is shown to perform well over a range of parameters, including changes in demand. The model, analysis and evaluation are then generalized to an arbitrary number, M, of job types.

Next, the problem of how best to structure and control a distributed computer system containing many processors is considered. The performance trade-offs associated with different tree structures are evaluated approxi-

mately by applying appropriate queueing models. It is shown that, for a given set of parameters and job distribution policy, there is an optimal tree structure that minimizes the overall average response time. This is obtained numerically through comparison of average response times. A simple heuristic policy is shown to perform well under certain conditions.

The last model addresses the trade-offs between reliability and performance. A number of servers, each of which goes through alternating periods of being *operative* and *inoperative*, offer services to an incoming stream of demands. The objective is to evaluate and optimize performance and cost metrics. A large real-life data set containing information about server breakdowns is analyzed first. The results indicate that the durations of the operative periods are not distributed exponentially. However, hyperexponential distributions are found to be a good fit for the observed data. A model based on these distributions is then formulated, and is solved exactly using the method of spectral expansion. A simple approximation which is accurate for heavily loaded systems is also proposed. The results of a number of numerical experiments are reported.

Declaration

All work contained within this thesis represents the original contribution of the author. Most of the material in this thesis has been, or will shortly be, published in conference proceedings and a journal, as listed below. The material in chapter 2 has been published in 1; some of the material in chapter 3 has been published in 2, 3 and 6; some of the material in chapter 4 has been published in 4 and 5; the material presented in chapter 5 is to be published in 7.

1. Dynamic Server Allocation in Heterogeneous Clusters, Palmer, J. and Mitrani, I. In *Proc First International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks*(HETNETs 2003) Kouvatsos, D. (ed) pp. 12/1-12/10, 2003

2. Dynamic Allocation of Servers in a Grid Hosting Environment, Fisher, M., Kubicek, C., McKee, P. Mitrani, I., Palmer, J. and Smith, R. In *Proc Fifth IEEE/ACM International Workshop on Grid Computing* (Grid 2004), USA, 2004, Buyya, R. (ed), pp. 421-426, IEEE Computer Society, 2004

3. Optimal Server Allocation in Reconfigurable Clusters with Multiple

Job Types, Palmer, J. and Mitrani, I. In *Proc Computational Science and its Applications* (ICCSA 2004), Italy, 2004. Pt. 2 Lagana, A. et al (eds), Lecture Notes in Computer Science Vol 3044 pp. 76-86, Springer, 2004

4. Optimal Tree Structures for Large-Scale Grids, Palmer, J. and Mitrani, I. In *Proc All Hands 2004, UK eScience Conference*, UK, 2004

5. Optimal Tree Structures for Large Service Networks, Palmer, J. and Mitrani, I. In *Proc 1st EuroNGI Conference on Next Generation Internet* (NGI 2005), Italy, 2005. IEEE Computer Society, 2005

6. Optimal and Heuristic Policies for Dynamic Server Allocation, Palmer, J. and Mitrani, I. *J. Parallel Distrib. Computing*, Vol 65/10 pp. 1204-1211, Elsevier, 2005

7. Empirical and Analytical Evaluation of Systems with Multiple Unreliable Servers, Palmer, J. and Mitrani, I. to be presented at PDS 2006, Philadelphia (in conjunction with DSN 2006)

Acknowledgements

I would like to express my sincere gratitude to the following people who have helped me in the completion of this thesis.

First and foremost, I would like to thank my supervisor, Professor Isi Mitrani for his guidance and support over the last four years.

Some of the research within this thesis was initiated as part of the collaborative GridSHED project. My thanks go to British Telecom and to the North-East Regional e-Science Centre (United Kingdom) for providing this project's financial support. I would also like to thank Sun Microsystems, who provided the data set used in chapter 5.

Thanks are also due to many of the staff here in the School of Computing Science at the University of Newcastle. The environment within the school has consistently been one of encouragement and support. Particular mention goes to Professor Paul Watson, following whose advice I decided to embark on my Ph.D. studies, and to Professor Santosh Shrivastava for believing me capable of postgraduate study in the first place.

Finally, I would like to thank my husband, Doug, for his understanding and encouragement.

Contents

Abstract i

Declaration iii

Acknowledgements v

Glossary xiii

1 Introduction 1

1.1 Motivation 1

1.2 Related Work 6

1.2.1 Server Allocation 6

1.2.2 Dynamic Provisioning 12

1.2.3 Network Configuration 15

1.2.4 Breakdowns and Repairs 16

2 Dynamic Server Allocation: Two Job Types **17**

2.1 Introduction 17

2.2 The Model 18

2.3 Computation of the Optimal Policy 21
2.4 Experimental results 24
2.5 Heuristic Policies 30
2.6 Conclusions 32

3 Dynamic Server Allocation: M Job Types **34**

3.1 Introduction 34
3.2 The Model 35
3.3 Computation of the Optimal Policy 37
3.4 Experimental results 41
3.5 Heuristic Policies 45
3.6 Conclusions 50

4 Optimal Tree Structures **51**

4.1 Introduction 51
4.2 Two-Level Tree Structures 53
4.2.1 The Model 53
4.2.2 Computation of The Optimal Tree Structure 57
4.2.3 A simple heuristic 59
4.2.4 Results 60
4.2.5 Conclusions 63
4.3 M-Level Tree Structures 64
4.3.1 The Model 64
4.3.2 Computation of the Optimal Tree Structure 67
4.3.3 A simple heuristic 71

4.3.4 Results 73
4.3.5 Conclusions 80

5 Breakdowns and Repairs **81**
5.1 Introduction 81
5.2 The data set 84
5.3 The model and its solution 90
5.3.1 Spectral expansion solution 94
5.3.2 A simple approximation 98
5.4 Numerical results 99
5.5 Conclusions 104

6 Conclusions and Future Work **106**
6.1 Future Work 110

Bibliography **113**

List of Figures

1.1 A service provisioning system with local and global resources . 3

1.2 Computing resources allocated into conceptual pools 5

1.3 Computing resources configured into conceptual network trees 6

2.1 Two reconfigurable heterogeneous clusters 19

2.2 Policy comparisons: increasing N 32

2.3 Policy comparisons: increasing loads 33

3.1 M reconfigurable heterogeneous clusters 35

3.2 Policy comparisons: $M = 3$ and increasing N 48

3.3 Policy comparisons: $M = 3$ and increasing loads 49

4.1 Flat structure 54

4.2 Service cluster split into k sub-clusters 54

4.3 Performance of different configurations 61

4.4 Optimal and heuristic configurations 62

4.5 Optimal and heuristic configurations 62

4.6 A tree with three levels of master nodes 65

4.7 Comparison of different routing policies: $m = 2$ 74

4.8 Comparison of theory and simulation: $m = 2$ 75

4.9 Comparison of theory and simulation: $m = 3$ 76

4.10 Performance of different configurations: $m = 3$ 77

4.11 Performance of different configurations: $m = 3$ 78

4.12 Performance of different configurations: $m = 3$ 79

5.1 A multi-server system with breakdowns and repairs 82

5.2 Alternating Periods of availability and unavailability 84

5.3 Densities of operative periods $(0 - 250)$ 89

5.4 Cumulative distribution of operative periods 89

5.5 Densities of Inoperative periods $(0 - 1.2)$ 91

5.6 Cumulative distribution of inoperative periods 91

5.7 Cost as a function of N 101

5.8 Average queue size against coefficient of variation 102

5.9 Average queue size against average repair time 103

5.10 Exact and approximate solutions: increasing load 103

5.11 Average response time as a function of N 105

List of Tables

2.1 Optimal actions: zero switching times, $N = 2$, $k_1 = 1$, $\lambda_1 = \lambda_2 = 0.086$, $\mu_1 = \mu_2 = 0.207$, $c_1 = 1$, $c_2 = 2$, $c_{1,2} = c_{2,1} = 10.0$. 25

2.2 Optimal actions: zero switching times, $N = 2$, $k_1 = 0$, $\lambda_1 = \lambda_2 = 0.086$, $\mu_1 = \mu_2 = 0.207$, $c_1 = 1$, $c_2 = 2$, $c_{1,2} = c_{2,1} = 10.0$. 26

2.3 Optimal actions: zero switching times, $N = 2$, $k_1 = 2$, $\lambda_1 = \lambda_2 = 0.086$, $\mu_1 = \mu_2 = 0.207$, $c_1 = 1$, $c_2 = 2$, $c_{1,2} = c_{2,1} = 10.0$. 26

2.4 Optimal actions: non-zero switching times, $N = 2$, $k_1 = 1$, $\lambda_1 = \lambda_2 = 0.047$, $\mu_1 = \mu_2 = 0.113$, $c_1 = 1$, $c_2 = 2$, $\xi = \eta = 0.113$ 27

2.5 Optimal actions: non-zero switching times, $N = 2$, $k_1 = 0$, $\lambda_1 = \lambda_2 = 0.047$, $\mu_1 = \mu_2 = 0.113$, $c_1 = 1$, $c_2 = 2$, $\xi = \eta = 0.113$ 28

2.6 Optimal actions: non-zero switching times, $N = 2$, $k_1 = 2$, $\lambda_1 = \lambda_2 = 0.047$, $\mu_1 = \mu_2 = 0.113$, $c_1 = 1$, $c_2 = 2$, $\xi = \eta = 0.113$ 28

3.1 Optimal actions: non-zero switching times, $N = 3$, $M = 3$, $k_1 = k_2 = k_3 = 1$, $j_1 = 0$, $\lambda_1 = \lambda_2 = \lambda_3 = 0.111$, $\mu_1 = \mu_2 = \mu_3 = 0.111$, $c_1 = 2$, $c_2 = c_3 = 1$, $\zeta_{i,j} = 0.111 (i \neq j)$ 43

3.2 Optimal actions: non-zero switching times, $N = 3$, $M = 3$, $k_1 = 1$, $k_2 = 2$, $k_3 = 0$, $j_1 = 0$, $\lambda_1 = \lambda_2 = \lambda_3 = 0.111$. $\mu_1 = \mu_2 = \mu_3 = 0.111$, $c_1 = 2$, $c_2 = c_3 = 1$, $\zeta_{i,j} = 0.111 (i \neq j)$. 43

3.3 Optimal actions: non-zero switching times, $N = 4$, $M = 3$, $k_1 = 1$, $k_2 = 2$, $k_3 = 1$, $j_1 = 1$, $\lambda_1 = \lambda_2 = \lambda_3 = 0.0909$, $\mu_1 = \mu_2 = \mu_3 = 0.0909$, $c_1 = 2$, $c_2 = c_3 = 1$, $\zeta_{i,j} = 0.0909 (i \neq j)$ 45

4.1 Optimal and heuristic configurations: $m = 3, N = 200$ 80

Glossary

λ, λ_i	Arrival rate, for job type i where specified
μ_i, ν	Service rate, for job type i where specified
ρ_i	Offered load, equal to λ_i/μ_i
$\xi, \eta, \zeta_{i,j}$	Switching rate of servers, between job types i and j where specified
$c_{i,j}$	Cost of switching a server from job type i to job type j
$m_{i,j}$	Number of switches currently in progress from job type i to job type j
j_i	Number of jobs of type i present
N	Total number of servers
M	Total number of job types
c_i	Holding cost of job type i (chapters 2 and 3) Service rate constant of proportionality at level i (chapter 4)
k_i	Number of servers currently serving job type i (chapters 2 and 3) Number of master nodes at level $i+1$ of the tree (chapter 4)
T_i	Average transfer time from level i to level $i+1$
W_i	Average sojourn time at a master node at level i
S_m	Average sojourn time at a service sub-cluster at level m
W	Average response time
L	Average queue length
α_i, β_i	Weight of operative/inoperative phase i of hyperexponential distribution
ξ_i, η_i	Rate of operative/inoperative phase i of hyperexponential distribution

Chapter 1

Introduction

1.1 Motivation

This thesis is motivated by recent developments in distributed processing, and in particular by the emerging concept of *hosting* or *service provisioning* systems. Advances in high performance hardware and the widespread availability of ultra-fast network access have enabled the connection of previously isolated computing resources, giving users access to remote servers with computing and storage capabilities which may be positioned anywhere on a local or wide network. For example, a *Computing Grid* is defined as the technology enabling the coupling of such resources which may be both geographically and administratively dispersed. Hence, heterogeneous clusters of servers may provide a variety of services to widely distributed user communities. It is proposed to determine strategies for maximizing the efficiency in hosting environments which may be widely distributed, as in the example of a Computing Grid, or locally distributed within a service cluster.

A more *efficient* environment is defined as one which makes the best possible use of all available resources. This may be analysed using performance measures such as the average number of waiting jobs present in the system or, following a request for service, the average response time. The latter is defined as the elapsed time between the arrival of the service request and the completed service. Greater efficiency is analogous to fewer waiting jobs or a faster response time.

An example of a service provisioning system is illustrated in figure 1.1. Demands for service arrive into a management system which is responsible for the allocation of service requests or *jobs* to available computing resources. The management system may also be responsible for the structuring and maintenance of the computing resources. These resources may be distributed across a very wide area network (possibly global) or may be grouped into a local cluster.

A static allocation of computing resources is likely to lead to poor utilization within the provisioning system. The random nature of user demand may lead to certain services being over-subscribed, and hence increased queue lengths would lead to longer response times. Other available resources may at the same time be under-utilized; these may be more efficiently allocated elsewhere. Similarly, a static configuration of the underlying network structure may not always be efficient. The speed of routing decisions within a large network of available resources may vary with time. Also, the number of choices between possible destinations for a service request will vary whenever the amount of available resources changes. Both of these factors will affect the efficiency of a particular network configuration.

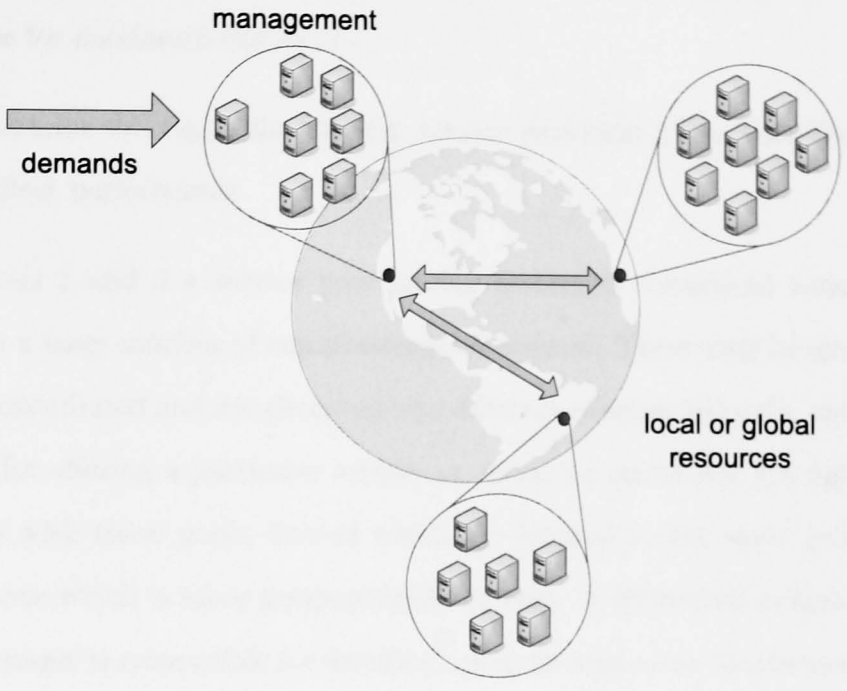

Figure 1.1: A service provisioning system with local and global resources

How many servers to allocate to a particular service in the light of reliability issues is another interesting problem when aiming to maximize efficiency. On the one hand, enough servers need to be provided to ensure a certain quality of service. However, knowledge of the minimum number of servers necessary to guarantee a desired level of performance would be desirable.

Hence, the objectives of this thesis include:

1. to determine policies for the dynamic allocation of servers to a particular service or job type as demand changes;

2. to determine optimal topological structures of available computing resources for maximum efficiency;

3. to determine strategies for efficient service provision when reliability may affect performance.

In chapters 2 and 3 a service provisioning system is considered which may contain a large number of computational resources. These may be geographically distributed and are allocated into different conceptual pools, each responsible for offering a particular *service* or serving a particular *job type*. An example with three pools, two of which are located in the same local cluster and one which is more geographically distant, is illustrated in figure 1.2. The manager is responsible for deciding upon the allocation of resources between pools. Requests for a particular service or job type are routed to the appropriate pool. A static allocation of resources to service pools is likely to lead to over-utilization of some resources and under-utilization of others. Strategies for *dynamic* allocation of resources are investigated.

In chapter 4 the *structure* of a large set of computing resources in considered. The performance trade-offs associated with different tree structures are evaluated. Two example tree structures are shown in figure 1.3. A manager at a particular level is responsible for deciding which is the best possible route for a particular service request or job traversing the tree. Strategies for deciding upon the most efficient tree structure for a particular environment or system are explored.

In chapter 5 the effect that server breakdowns and other outages have on the performance of a service provisioning system is investigated. This

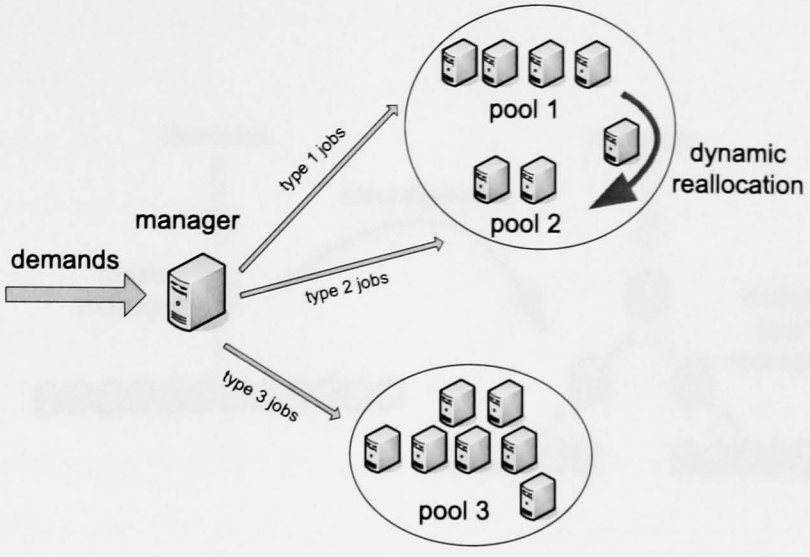

Figure 1.2: Computing resources allocated into conceptual pools

research was initiated following discussions with Sun Microsystems, who provided a large real-life data set for analysis. Conclusions and future work are presented in chapter 6.

The contribution of this thesis is in the analysis, evaluation and optimization of a variety of operating strategies for large scale service provisioning systems. The novelty of the work resides in its inclusion of (a) models with many servers, (b) dynamic decision making and (c) realistic treatment of breakdowns and repairs.

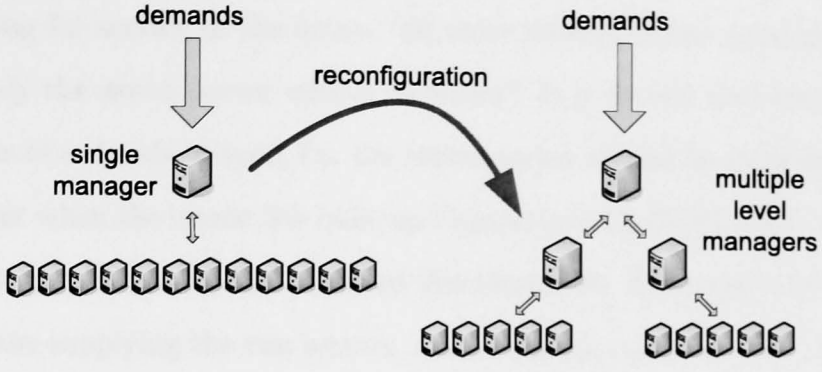

Figure 1.3: Computing resources configured into conceptual network trees

1.2 Related Work

1.2.1 Server Allocation

Despite an extensive literature on dynamic optimization (some good general texts are [1, 43, 47]), the multi-server problem addressed within chapters 2 and 3 does not appear to have been studied before.

There was some preliminary research into the optimal control of queueing systems with two heterogeneous servers by Lin and Kumar [27]. They address the system of a single queue of Poisson arrivals served by two heterogeneous servers. Both servers have exponentially distributed service times, although these may be with different means. The aim is to minimize the mean response time of customers, which by Little's theorem is equivalent to

minimizing the mean number of customers in the system. It is shown that if the faster server is idle, then it is optimal to supply it with a customer if one is waiting for service in the queue. Of more interest is the question: when to supply the slower server with a customer? It is proven that the optimal policy is of a threshold type, i.e. the slower server should be supplied with a customer when the queue has built up beyond a computable level. An algorithm is supplied to readily calculate this threshold. This work only models one queue supplying the two servers, and switching costs are zero. It is surmised that this threshold policy will not be generalised to the multi-server case, when the threshold for each server may be dependent on which other servers are busy.

There is also a body of work on optimal allocation in the context of polling systems, where a server visits several queues in a fixed or variable order, with or without switching overheads.

Hofri and Ross [19] consider the problem of two unbounded queues fed by Poisson arrival processes which are served by a single server. The service may not be interrupted, and when the server wants to switch from one queue to the other, a setup time is incurred. It is assumed that the service time distribution is the same for each queue. The goal of this paper is to find the optimal non-preemptive server assignment policy which minimizes the sum of the holding costs and switching charges.

For both the discounted cost and the long-run average cost criterion, it is found that the optimal policy is exhaustive, i.e. if the server is at a given queue, it is best to remain there at least until that queue is emptied. It

remains to specify when to reassign a server at an empty queue. Here, it is found that the server should remain in this queue until the other queue length achieves or exceeds a pre-determined threshold. The policy is then specified by these two thresholds, one for each queue. These threshold values are found by numerical analysis as a function of system parameters. It is suspected that this threshold policy in the case with more queues cannot be captured in a simple characterization.

Duenyas and Van Oyen [15] investigate the allocation of a single server to a system of queues with set-up times when switching the server from one queue to another. Each queue represents a class of jobs and possesses a holding cost rate. The services have a general distribution as do the set-ups. The objective of the research is to minimize the average holding cost per unit time. The special case of zero switching times corresponds to the $c\mu$-rule [4] mentioned below.

This is considered within the paper as a difficult problem, and the optimal policy is only partially characterized. However, a very good heuristic is introduced first for the case of 2 queues and further developed for the general case of N queues. This heuristic cannot be compared to an optimal policy because this has not been formulated. Instead, the heuristic is compared in simulation runs with other widely used policies in the literature, such as exhaustive, threshold policies and the $c\mu$-rule [4]. The heuristic is found to substantially outperform those other policies to which it is compared in many different scenarios. The optimal policy is not characterized or approximated by numerical methods

In a later paper, Duenyas and Van Oyen [14] investigate the allocation of a single server to a system of queues with switching costs incurred when switching the server instantaneously from one queue to another. This is again considered to be a difficult problem, and the optimal policy is also only partially characterized. A very good heuristic is introduced first for the case of 2 queues and further developed for the general case of N queues.

A numerical study is made comparing the heuristic to other policies within the literature (exhaustive policies, gated policies and the $c\mu$-rule [4]). The optimal policy is computed numerically with a truncated state space. This is computationally extremely expensive, and the heuristic is compared to the optimal pure Markov policy only for systems with 2 or 3 queues. The heuristic is found to compare favourably with the numerically computed optimal policy, and to outperform the other policies from the literature. This is an unlikely model with instantaneous switching and yet non-zero switching costs. The optimal policy is not characterized fully.

Koole [23] models a single server assigned to two heterogeneous queues, with the objective of minimizing holding costs and switching costs. The optimal policy for the preemptive dynamic assignment of the server to the queues with respect to long-run discounted or average costs is partially characterized.

The policy is computed numerically using dynamic programming techniques for a truncated state-space. It does not appear that the policy will be easy to describe. Switching away from the queue with the highest value of $c\mu$ only occurs if that queue is empty (i.e. it is served exhaustively once

the server is there, as also found by Hofri and Ross [19]). Switching to the queue which would have had higher priority under the $c\mu$-rule occurs when the length of this queue has reached some threshold level. This is intuitively because switching to this queue will reduce holding costs at a faster rate than remaining in the current queue, but must also overcome the switching costs. Hence, a certain threshold must be passed to justify the switch. It is proven that a threshold-like policy is optimal for a large enough queue length at the server with the lower value of $c\mu$.

This threshold policy is computed numerically and expressed in the form of a table for look-up. The optimal policy is compared to the threshold policies in simulation runs comparing long-run average costs. It is found that as switching costs increase, the $c\mu$ rule performs worse and the threshold policy performs better. Indeed, the threshold policy is quite close in performance to the optimal policy. However, switching times are instantaneous, whereas switching costs are non-zero. This is an unlikely model and the optimal policy is not characterized fully.

Results for optimal policies of various queueing and resource sharing models are again studied by Koole [24]. This paper models optimal admission control with geometric arrivals, which is a different problem to that with unbounded queue lengths. Customers may be rejected with associated costs and these together with holding costs are to be minimized. A threshold policy is again proven to be optimal for the discrete time model of one server, one queue. Most results found here by Koole are for a single server - however this is extended to the extra server of Lin and Kumar [27] mentioned earlier.

Liu, Nain and Towsley [29] prove some important results for polling systems. Again, the system is modelled as N queues attended by a single server. Switch over times are strictly positive, and the aim is to find polling policies that stochastically minimize the unfinished work and the number of customers in the system at all times. The optimal policy is proven to be both greedy and exhaustive, that is when the server is at a non-empty queue, it should neither idle nor switch until that queue is empty. Also, in the special case where the polling system is symmetric (arrival patterns and service time distributions are stochastically identical at all queues) the routing policy which moves the server to the queue with the largest queue length is optimal. If not all the queue lengths are known, a cyclic routing policy is optimal if the only information available is that of the previous decision. If in this special symmetric case the server empties a queue, the optimal policy is patient, i.e. the server always stays idling at the last visited queue whenever the system is empty. Important and interesting as these results for the polling system are, they do not address the issue of differing priority levels for different queues implied by variable unit storage costs. Holding costs in one queue are equal per job per unit time as at any other queue.

To summarize the polling literature, even in those cases of a single server, it has been observed by both Duenyas and Van Oyen [15, 14], and Koole [23, 24], that the presence of non-zero switching times makes the optimal policy very difficult to characterize explicitly. This necessitates the consideration of heuristic policies.

The only general result available for multiprocessor systems applies when the switching times and costs are zero: then the $c\mu$-rule is optimal, i.e. the best policy is to give absolute preemptive priority to the job type for which the product of holding cost and service rate is largest. Buyukkoc et al [4] show this rule to be optimal for arbitrary arrival processes when storage costs are zero, provided that the service times are geometric and the service discipline is preemptive. The $c\mu$-rule is work-conserving i.e. it is not optimal to keep a server idle when customers are present and not being served.

A model similar to the one presented in this thesis was analyzed by Fayolle et al [16]. There the policy is fixed (servers are switched instantaneously, and only when idle), and the object is to evaluate the system performance. The solution is complex and rather difficult to implement. The more realistic model of multi-server clusters and non-zero switching times is not addressed.

1.2.2 Dynamic Provisioning

Following publication of the work presented here in chapters 2 and 3 on dynamic server allocation [40, 41], a body of work has appeared in the area of dynamic provisioning. Chase et al. [6] consider the possibility of switching servers between services to cope with demand. They propose an ad-hoc policy which does not take either costs or QoS requirements into account. A cluster management architecture is described called *Cluster-on-Demand* (COD). COD controls local resources, and dynamic policy-based allocation

of cluster resources across different application environments is made possible (i.e., across different offered services or jobs). Experimental results are presented to demonstrate simple priority allocation policies (allocating resources to higher priority clusters whenever necessary, although a guaranteed minimum may be specified for the number of nodes allocated to any specific request type). The paper concentrates on the *mechanisms* for dynamic resource management rather than the *policies* used. An optimal policy for the dynamic provisioning is not discussed, and hence there is no benchmark for the performance results other than as an improvement upon a system without any dynamic reallocation of resources.

More recently, several companies have introduced products to the market which offer dynamic provisioning in order to increase resource utilization. Of particular interest are the systems described by Sun Microsystems and IBM. Although details of the dynamic resource allocation are limited, these represent the importance of policy-based management of resources in a Grid hosting environment.

Sun Microsystems describe their N1 Grid System [31] as an architecture capable of delivering what is referred to as *Just in Time Computing*. This is in essence a grid hosting environment which includes the flexibility of dynamic and automatic resource provisioning, enabling service level management. The importance of such flexibility is highlighted as having the potential to transform the way in which large clusters of computing resources are managed. The details of any policies used for such dynamic provisioning of resources are omitted.

IBM have released a provisioning on demand system known as the IBM

Tivoli Intelligent ThinkDynamic Orchestrator [30]. This promises to increase resource utilization and achieve dynamic infrastructure allocation. Resources can be allocated and managed dynamically without intervention through an architectural layer called *Policy-Based Orchestration.* Different business policies may be defined and the provisioning of resources will respond dynamically accordingly as conditions change. It is stated that peaks in demand are sensed automatically and responded to by allocating resources to the most important processes based on a business policy. However, once again, details of policies or heuristics used are omitted.

Clark et al. [7] describe the possibility of migrating operating system (OS) instances across distinct physical hosts. The majority of the migration is carried out while operating systems continue to run, and impressive performance is obtained for service downtimes, in the order of magnitude of milliseconds. Hence, live migration of operating systems is shown to be a practical tool. The paper is concerned mostly with the implementation of this live OS migration, demonstrating its viability. However, an application of the described work is within cluster management: there are opportunities for dynamic load balancing of processor resources. A challenge is highlighted within the paper to develop cluster control software capable of making informed decisions as to the placement and movement of virtual machines. The heuristics described here in chapters 2 and 3 could be applied for such a decision making process.

1.2.3 Network Configuration

The problem of load balancing across a network of available resources has been discussed in distributed systems literature for more than two decades. A comprehensive discussion of diffusion techniques for dynamic load balancing can be found in [8]. The objective of load balancing is to enable each available resource to perform an even share of the network load. Work is quantified in terms of *tasks*, each of which require an amount of processing time to be completed. Tasks may be reallocated from one processor to another, balancing the load across multiple servers. The aim is to minimise the overall execution time, using specified load balancing algorithms. A description of customized load balancing strategies for a network of workstations is given by Zaki et al [48]. Hine and Holzer [18] present their results on different scheduling algorithms for load balancing. More recently, Houle et al [20] consider algorithms for static load balancing on trees, assuming that the total load is fixed.

Koole [11, 12] describes some recent work on dynamic load balancing in the context of parallel applications running in a grid environment. The first paper presents the impact of fluctuations in processing speeds on running times in a grid environment, while the second paper presents results of extensive load-balancing experiments. In particular, the research focuses on fluctuations in processing speeds and the associated challenge of balancing the load of dependent iterations across heterogeneous processors. Forecasts of processor speed are obtained via the method of exponential smoothing. Dynamic load balancing decisions are made whereby the number of rows of

a parallel application (which could be considered to be jobs) assigned to any particular processor is proportional to its predicted processor speed.

In all of these studies, the network configuration is treated as fixed and immutable. Instead, the approach outlined in this thesis considers the dynamic reconfiguration of the underlying tree structure as load changes, rather than the reallocation of jobs across a fixed tree. This does not appear to have been studied before.

1.2.4 Breakdowns and Repairs

When considering the addition of server breakdown to a model of a grid provisioning system for analysis, perhaps the simplest model to analyse is one where all servers are statistically identical, all breakdowns and repairs are independent of each other, and the operative and inoperative periods are distributed exponentially. Indeed, some results are already available for that special case (see [35]).

An example of a generalization where breakdowns and repairs are not necessarily independent of each other, but still may occur at exponentially distributed intervals, is presented in [5]. Having specified a model of interest, programs are provided that solve this exactly. These can then be invoked with different parameters for purposes of optimisation. However, models in which the breakdowns and repairs occur at intervals which are non-exponentially distributed, or models of the size and complexity as here described in chapter 5 have not previously been considered.

Chapter 2

Dynamic Server Allocation: Two Job Types

2.1 Introduction

The provision of a Grid service involves the hosting of heterogeneous clusters of servers. These may provide a variety of services to widely distributed user communities. Users submit jobs without necessarily knowing, or caring, where they will be executed. The system distributes those jobs among the servers, attempting to make the best possible use of the available resources and provide the best possible quality of service.

The random nature of user demand, and also changes of demand patterns over time, can lead to temporary oversubscription of some services and underutilization of others. In such situations, it could be advantageous to reallocate servers from one type of provision to another, even at the cost of switching overheads. The question that arises in that context is how to

decide whether, and if so when, to perform such reconfigurations.

Posed in its full generality, this is a complex problem which is most unlikely to yield an exact and explicit solution. This first approach is to examine a simple, yet non-trivial special case, where the optimal dynamic reallocation policy can be computed numerically. Some heuristic policies are then proposed which, while not optimal, perform reasonably well and are easily implementable. The quality of the heuristics, compared to the optimal policy, is evaluated by simulation.

The system under consideration contains a pool of N servers, split into two heterogeneous clusters of sizes K and $N-K$ respectively. Cluster 1 is dedicated to a queue of jobs of type 1 (e.g., short web accesses), while cluster 2 serves a queue of jobs of type 2 (e.g., long database searches). Type 1 jobs have different response time requirements (e.g., they are less tolerant of delays) than type 2. It is possible to reassign any server from one queue to the other, but the process is generally not instantaneous and during it the server becomes unavailable. In those circumstances, a reconfiguration policy would specify, for any given parameter set (including costs), and current state, whether to switch a server or not.

2.2 The Model

The model is illustrated in Figure 2.1. Jobs of type i arrive according to an independent Poisson process with rate λ_i, and join a separate unbounded queue ($i = 1, 2$). Their required service times are distributed exponentially with mean $1/\mu_i$. The cost of keeping a type i job in the system is c_i per

unit time ($i = 1, 2$). These 'holding' costs reflect the relative importance, or willingness to wait, of the two job types.

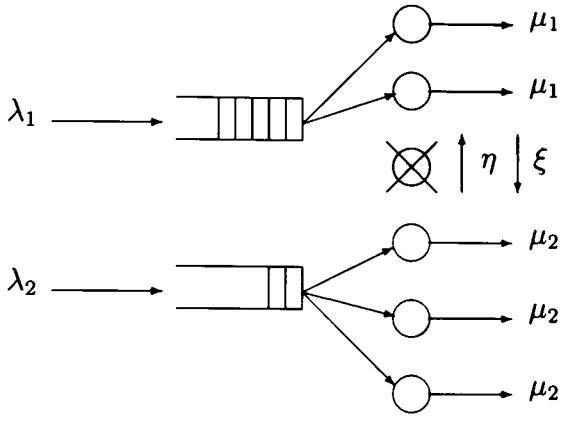

Figure 2.1: Two reconfigurable heterogeneous clusters

Any server currently allocated to queue 1 may be switched to queue 2. Such a switch costs $c_{1,2}$ and takes an interval of time distributed exponentially with mean $1/\xi$, during which the server cannot serve jobs. Similarly, a server allocated to queue 2 may be switched to queue 1, at the cost of $c_{2,1}$ and taking an interval of time distributed exponentially with mean $1/\eta$. It is assumed that switches are initiated at job arrival or departure instants. Indeed, it is at those instants that switches may become advantageous, and if they do, they should be performed without delay. Also, it is assumed that the switching policy employed is memoryless, i.e., switching decisions may depend on the current state but not on past history.

Any job whose service is interrupted by a switch returns to the appropriate queue and resumes service from the point of interruption when a server becomes available for it.

The system state at any time is described by a quintuple of integers,

$S = (j_1, j_2, k_1, m_{1,2}, m_{2,1})$, where j_i is the number of type i jobs present $(i = 1, 2)$, k_1 is the current number of servers allocated to queue 1, $m_{1,2}$ is the number of servers currently being reallocated from queue 1 to queue 2, and $m_{2,1}$ is the number of servers currently being reallocated from queue 2 to queue 1. Only states satisfying $k_1 + m_{1,2} + m_{2,1} \leq N$ are valid. The number of servers currently allocated to queue 2 is equal to $k_2 = N - k_1 - m_{1,2} - m_{2,1}$.

Under the above assumptions, the system is modelled by a continuous time Markov process. The transition rates of that process depend on the switching policy, i.e. on the decisions (actions) taken in various states. Denote by $r_d(S, S')$ the transition rate from state S to state S' $(S \neq S')$, given that action d is taken. The possible actions are (a) do nothing, (b) initiate a switch from queue 1 to queue 2 (if $k_1 > 0$) and (c) initiate a switch from queue 2 to queue 1 (if $k_2 > 0$). These actions are represented by $d = 0$, $d = 1$ and $d = 2$, respectively.

The values of $r_d(S, S')$, for $S = (j_1, j_2, k_1, m_{1,2}, m_{2,1})$ and $d = 0$, are given by

$$r_0(S, S') = \begin{cases} \lambda_1 & \text{if } S' = (j_1 + 1, j_2, k_1, m_{1,2}, m_{2,1}) \\ \lambda_2 & \text{if } S' = (j_1, j_2 + 1, k_1, m_{1,2}, m_{2,1}) \\ \min(j_1, k_1)\mu_1 & \text{if } S' = (j_1 - 1, j_2, k_1, m_{1,2}, m_{2,1}) \\ \min(j_2, k_2)\mu_2 & \text{if } S' = (j_1, j_2 - 1, k_1, m_{1,2}, m_{2,1}) \\ m_{1,2}\xi & \text{if } S' = (j_1, j_2, k_1, m_{1,2} - 1, m_{2,1}) \\ m_{2,1}\eta & \text{if } S' = (j_1, j_2, k_1 + 1, m_{1,2}, m_{2,1} - 1) \\ 0 & \text{otherwise} \end{cases}$$

The corresponding rates for $d = 1$ are obtained by replacing, in S', k_1 by $k_1 - 1$ and $m_{1,2}$ by $m_{1,2} + 1$. Similarly, the rates for $d = 2$ are obtained by replacing $m_{2,1}$ by $m_{2,1} + 1$. Note that, in cases $d = 1$ and $d = 2$, there is a zero-time transition which changes k_1 or k_2, and then an exponentially distributed interval with mean $1/r_d(S, S')$, after which the state jumps to S'.

The total transition rate out of state S, given that action d is taken, $r_d(S)$, is equal to:

$$r_d(S) = \sum_{S'} r_d(S, S') .$$

2.3 Computation of the Optimal Policy

For the purposes of optimization, it is convenient to apply the technique of uniformization to the Markov process (e.g., see [10]). This entails the introduction of 'fictitious' transitions which do not change the system state, so that the average interval between consecutive transitions ceases to depend on the state, and then embedding a discrete-time Markov chain at transition instants. First, find a constant, Λ, such that $r_d(S) \leq \Lambda$ for all S and d. A suitable value for Λ is

$$\Lambda = \lambda_1 + \lambda_2 + N(\mu_1 + \mu_2 + \xi + \eta) . \tag{2.1}$$

Next, construct a Markov chain whose one-step transition probabilities

when action d is taken, $q_d(S, S')$, are given by

$$q_d(S, S') = \begin{cases} r_d(S, S')/\Lambda & \text{if } S' \neq d(S) \\ 1 - r_d(S)/\Lambda & \text{if } S' = d(S) \end{cases}$$

where $d(S)$ is the state resulting from the immediate application of action d in state S. This Markov chain is, for all practical purposes, equivalent to the original Markov process.

Without loss of generality, the unit of time can be scaled so that the uniformization constant becomes $\Lambda = 1$.

The finite-horizon optimization problem can be formulated as follows. Denote by $V_n(S)$ the minimal expected total cost incurred during n consecutive steps of the Markov chain, given that the current system state is S. The cost incurred at step l in the future is discounted by a factor α^l ($l = 1, 2, \ldots, n-1$; $0 \leq \alpha \leq 1$). Setting $\alpha = 0$ implies that all future costs are disregarded; only the current step is important. When $\alpha = 1$, the cost of a future step, no matter how distant, carries the same weight as the current one.

Any sequence of actions which achieves the minimal cost $V_n(S)$, constitutes an 'optimal policy' with respect to the initial state S, cost parameters, event horizon n, and discount factor α.

Suppose that the action taken in state S is d. This incurs an immediate cost of $c(d)$, equal to $c_{1,2}$ if $d = 1$ and $c_{2,1}$ if $d = 2$. In addition, since the average interval between transitions is 1, each type 1 job in the system incurs a holding cost c_1 and each type 2 job in the system incurs a holding cost c_2.

The next state will be S', with probability $q_d(S, S')$, and the minimal cost of the subsequent $n-1$ steps will be $\alpha V_{n-1}(S')$. Hence, the quantities $V_n(S)$ satisfy the following recurrence relations:

$$V_n(S) = j_1 c_1 + j_2 c_2 + \min_d \left[c(d) + \alpha \sum_{S'} q_d(S, S') V_{n-1}(S') \right] . \qquad (2.2)$$

Thus, starting with the initial values $V_0(S) = 0$ for all S, one can compute $V_n(S)$ in n iterations. In order to make the state space finite, the queue sizes are bounded at some level, $j_1 < J$, $j_2 < J$. Then, if $V_{n-1}(S)$ has already been computed for some n and for all S, the complexity of computing $V_n(S)$, for a particular state S, is roughly constant. Three actions need to be compared, and the best action to take in that state, and for that n, is indicated by the value of d that achieves the minimum in the right-hand side of (2.2). Since there are on the order of $O(J^2N^3)$ states altogether, the computational complexity of one iteration is on the order of $O(J^2N^3)$, and hence overall complexity of solving (2.2) and determining the optimal switching policy over a finite event horizon of size n, is on the order of $O(J^2N^3n)$.

If the discount factor α is strictly less than 1, it reasonable to consider the infinite-horizon optimization, i.e. the total minimal expected cost, $V(S)$, of all future steps, given that the current state is S. That cost is of course infinite when $\alpha = 1$, but it is finite when $\alpha < 1$. Indeed, in the latter case it is known (see [2]), that under certain rather weak conditions, $V_n(S) \to V(S)$ when $n \to \infty$. When the optimal actions depend only on the current state, S, and not on n, the policy is said to be 'stationary'.

An argument similar to the one preceding (2.2) leads to the following

equation for $V(S)$:

$$V(S) = j_1 c_1 + j_2 c_2 + \min_d \left[c(d) + \alpha \sum_{S'} q_d(S, S') V(S') \right] . \qquad (2.3)$$

The optimal policy (i.e. the best action in any given state) is specified by the value of d that achieves the minimum in the right-hand side of (2.3).

Equation (2.3) can be solved by performing the finite-horizon iterations, stopping when the difference in cost functions between two consecutive iterations becomes sufficiently small. Alternatively, if the policy is of greater interest than the cost function, the iterations may be stopped when the policy becomes stationary.

Equation (2.3) may also be solved by applying the 'policy improvement' algorithm [13]. This iterative process is later described for the generalized model in chapter 3.

2.4 Experimental results

For initial analysis, consider a simple system with $N = 2$, where switches cost money but do not take time. Although this case is not of great practical interest, it is included as an illustration. The system state is described by a triple, $S = (j_1, j_2, k_1)$. The number of servers allocated to type 2 is $k_2 = N - k_1$. The uniformization constant is now $\Lambda = \lambda_1 + \lambda_2 + N(\mu_1 + \mu_2)$. If action $d = 1$ or $d = 2$ is taken in state S, the value of k_1 changes immediately, and then a new state is entered after an exponentially distributed interval with mean $1/\Lambda$.

In this example, the arrival and service parameters of the two job types are the same, but waiting times for type 2 are twice as expensive as those for type 1. The discount factor is $\alpha = 0.95$. The stationary optimal policy for states where $k_1 = k_2 = 1$ is shown in table 2.1. The truncation level used in the computation was $J = 30$, but the table stops at $j_1 = j_2 = 10$; the actions do not change beyond that level.

		j_2										
		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	1	1	1	1	1	1	1	1
	1	0	0	0	0	0	1	1	1	1	1	1
	2	0	0	0	0	0	1	1	1	1	1	1
	3	0	0	0	0	0	1	1	1	1	1	1
	4	0	0	0	0	0	1	1	1	1	1	1
j_1	5	2	0	0	0	0	1	1	1	1	1	1
	6	2	0	0	0	0	1	1	1	1	1	1
	7	2	0	0	0	0	1	1	1	1	1	1
	8	2	0	0	0	0	1	1	1	1	1	1
	9	2	0	0	0	0	1	1	1	1	1	1
	10	2	0	0	0	0	1	1	1	1	1	1

Table 2.1: Optimal actions: zero switching times, $N = 2$, $k_1 = 1$, $\lambda_1 = \lambda_2 = 0.086$, $\mu_1 = \mu_2 = 0.207$, $c_1 = 1$, $c_2 = 2$, $c_{1,2} = c_{2,1} = 10.0$

As expected, the presence of switching costs discourages switching; a server is sometimes left idle even when there is work to be done. Note that the $c\mu$-rule in this case would give preemptive priority to type 2: it would take action $d = 1$ whenever $j_2 \geq 2$, and action $d = 2$ when $j_2 = 0$, $j_1 \geq 2$.

The optimal policy for $k_1 = 0$ (all servers currently allocated to job type 2) is shown in table 2.2. This table shows that it is now optimal to take action $d = 2$ when $j_1 = 1$ and $j_2 = 0$, or when $j_1 > 1$ and $j_2 < 2$. The optimal policy for $k_1 = 2$ (all servers currently allocated to job type 1) is

		j_2										
		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
	1	2	0	0	0	0	0	0	0	0	0	0
	2	2	2	0	0	0	0	0	0	0	0	0
	3	2	2	0	0	0	0	0	0	0	0	0
	4	2	2	0	0	0	0	0	0	0	0	0
j_1	5	2	2	0	0	0	0	0	0	0	0	0
	6	2	2	0	0	0	0	0	0	0	0	0
	7	2	2	0	0	0	0	0	0	0	0	0
	8	2	2	0	0	0	0	0	0	0	0	0
	9	2	2	0	0	0	0	0	0	0	0	0
	10	2	2	0	0	0	0	0	0	0	0	0

Table 2.2: Optimal actions: zero switching times, $N = 2$, $k_1 = 0$, $\lambda_1 = \lambda_2 = 0.086$, $\mu_1 = \mu_2 = 0.207$, $c_1 = 1$, $c_2 = 2$, $c_{1,2} = c_{2,1} = 10.0$

		j_2										
		0	1	2	3	4	5	6	7	8	9	10
	0	0	1	1	1	1	1	1	1	1	1	1
	1	0	1	1	1	1	1	1	1	1	1	1
	2	0	0	1	1	1	1	1	1	1	1	1
	3	0	0	1	1	1	1	1	1	1	1	1
	4	0	0	1	1	1	1	1	1	1	1	1
j_1	5	0	0	1	1	1	1	1	1	1	1	1
	6	0	0	1	1	1	1	1	1	1	1	1
	7	0	0	1	1	1	1	1	1	1	1	1
	8	0	0	1	1	1	1	1	1	1	1	1
	9	0	0	1	1	1	1	1	1	1	1	1
	10	0	0	1	1	1	1	1	1	1	1	1

Table 2.3: Optimal actions: zero switching times, $N = 2$, $k_1 = 2$, $\lambda_1 = \lambda_2 = 0.086$, $\mu_1 = \mu_2 = 0.207$, $c_1 = 1$, $c_2 = 2$, $c_{1,2} = c_{2,1} = 10.0$

shown in table 2.3. This table shows that it is now optimal to take action $d = 1$ when $j_1 \leq 1$ and $j_2 > 0$, or when $j_1 > 1$ and $j_2 > 1$.

From now on, models will be examined where switching takes non-zero time. To keep the number of parameters low, the monetary costs of switching will be assumed negligible, $c_{1,2} = c_{2,1} = 0$. The uniformization constant is given by (2.1), and the unit of time is chosen so that $\Lambda = 1$. Table 2.4 illustrates the stationary optimal policy when $k_1 = k_2 = 1$, for the same holding costs as in table 2.1.

		j_2										
		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	1	1	1	1	1	1	1	1	1
	1	0	0	0	0	0	0	1	1	1	1	1
	2	0	0	0	0	0	0	0	1	1	1	1
	3	0	0	0	0	0	0	0	1	1	1	1
	4	0	0	0	0	0	0	0	1	1	1	1
j_1	5	2	0	0	0	0	0	0	1	1	1	1
	6	2	0	0	0	0	0	0	1	1	1	1
	7	2	0	0	0	0	0	0	1	1	1	1
	8	2	0	0	0	0	0	0	1	1	1	1
	9	2	0	0	0	0	0	0	1	1	1	1
	10	2	0	0	0	0	0	0	1	1	1	1

Table 2.4: Optimal actions: non-zero switching times, $N = 2$, $k_1 = 1$, $\lambda_1 = \lambda_2 = 0.047$, $\mu_1 = \mu_2 = 0.113$, $c_1 = 1$, $c_2 = 2$, $\xi = \eta = 0.113$

Again, the observation is made that switching is discouraged, compared to the $c\mu$-rule, even though the average switching times are no larger than the average job service times.

The optimal policy for $k_1 = 0$ (all servers currently allocated to job type 2) is shown in table 2.5. It is now optimal to take action $d = 2$ when $j_1 > 0$ and $j_2 < 2$ or when $j_1 = j_2 = 0$. The optimal policy for $k_1 = 2$ (all servers

		j_2										
		0	1	2	3	4	5	6	7	8	9	10
	0	2	0	0	0	0	0	0	0	0	0	0
	1	2	2	0	0	0	0	0	0	0	0	0
	2	2	2	0	0	0	0	0	0	0	0	0
	3	2	2	0	0	0	0	0	0	0	0	0
	4	2	2	0	0	0	0	0	0	0	0	0
j_1	5	2	2	0	0	0	0	0	0	0	0	0
	6	2	2	0	0	0	0	0	0	0	0	0
	7	2	2	0	0	0	0	0	0	0	0	0
	8	2	2	0	0	0	0	0	0	0	0	0
	9	2	2	0	0	0	0	0	0	0	0	0
	10	2	2	0	0	0	0	0	0	0	0	0

Table 2.5: Optimal actions: non-zero switching times, $N = 2$, $k_1 = 0$, $\lambda_1 = \lambda_2 = 0.047$, $\mu_1 = \mu_2 = 0.113$, $c_1 = 1$, $c_2 = 2$, $\xi = \eta = 0.113$

		j_2										
		0	1	2	3	4	5	6	7	8	9	10
	0	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1
	2	0	0	1	1	1	1	1	1	1	1	1
	3	0	0	1	1	1	1	1	1	1	1	1
	4	0	0	1	1	1	1	1	1	1	1	1
j_1	5	0	0	0	1	1	1	1	1	1	1	1
	6	0	0	0	1	1	1	1	1	1	1	1
	7	0	0	0	1	1	1	1	1	1	1	1
	8	0	0	0	1	1	1	1	1	1	1	1
	9	0	0	0	1	1	1	1	1	1	1	1
	10	0	0	0	1	1	1	1	1	1	1	1

Table 2.6: Optimal actions: non-zero switching times, $N = 2$, $k_1 = 2$, $\lambda_1 = \lambda_2 = 0.047$, $\mu_1 = \mu_2 = 0.113$, $c_1 = 1$, $c_2 = 2$, $\xi = \eta = 0.113$

currently allocated to job type 1) is shown in table 2.6. It is now optimal to take action $d = 1$ when $j_1 < 2$, or $j_1 = 2, 3, 4$ and $j_2 > 1$, or $j_1 > 4$ and $j_2 > 2$.

In each of the above experiments the cost matrix was initialised to the holding cost of the current state. Approximately 150 iterations were required for the cost function to converge to within a tolerance level of 0.01. The calculations of the table look-ups for the optimal policy have been executed on a Windows machine with a 3GHz Intel Pentium 4 processor, and with 2GB of RAM. With 2 job types, available memory space was easily sufficient, even when storing the results from each iteration. The calculation time for each set of 3 tables was approximately 300ms.

The next question to be addressed is "How can one use dynamic optimization in practice?" Ideally, the optimal policy would be characterized explicitly in terms of the parameters, providing a set of rules to be followed (like, for example, the $c\mu$-rule). Unfortunately, such a characterization does not appear feasible for this problem.

Another approach is to pre-compute the optimal policy for a wide range of parameter values, and store a collection of tables such as table 2.1 and table 2.4. Then, having monitored the system and estimated its parameters, the optimal policy could be obtained by a table look-up. This is feasible, but may consume quite a lot of storage.

The third and most commonly used approach is to formulate a heuristic policy which (a) is simply characterized in terms of the parameters, and (b) performs acceptably well, compared with the optimal policy. That is what is proposed in the following section.

2.5 Heuristic Policies

A straightforward switching policy is to do no switching at all. Allocate the servers to the two queues roughly in proportion to the offered load, $\rho_i = \lambda_i/\mu_i$, and to the holding cost, c_i, for each type. In other words, set

$$k_1 = \left\lfloor N\frac{\rho_1 c_1}{\rho_1 c_1 + \rho_2 c_2} + 0.5 \right\rfloor \quad ; \quad k_2 = N - k_1 \ ,$$

if both k_1 and k_2 are non-zero, otherwise replace 0 by 1 and N by $N-1$. Having made the allocation, leave it fixed as long as the offered loads and costs remain the same. This will be referred to as the 'static' policy. It certainly has the virtue of simplicity, and also provides a comparator by which the benefits of dynamic reconfiguration can be measured.

The idea behind this dynamic heuristic policy is to attempt to balance the total holding costs of the two job types. That is, the policy tries to prevent the quantities $j_1 c_1$ and $j_2 c_2$ from diverging. The following rules are applied:

1. Take action $d = 1$ in state $S = (j_1, j_2, k_1, m_{1,2}, m_{2,1})$ if

$$c_1 \left\{ j_1 + \frac{1}{\xi}[\lambda_1 - \mu_1 \min(k_1 - 1, j_1)] \right\} > c_2 \left\{ j_2 + \frac{1}{\xi}[\lambda_2 - \mu_2 \min(k_2, j_2)] \right\}$$

2. Take action $d = 2$ in state $S = (j_1, j_2, k_1, m_{1,2}, m_{2,1})$ if

$$c_1 \left\{ j_1 + \frac{1}{\eta}[\lambda_1 - \mu_1 \min(k_1, j_1)] \right\} < c_2 \left\{ j_2 + \frac{1}{\eta}[\lambda_2 - \mu_2 \min(k_2 - 1, j_2)] \right\}$$

These rules are based on approximating the effects of a switch. If j_1 jobs of

type 1 are present and k_1 servers are available for them, then the average queue 1 increment during an interval of length x may be estimated as $x[\lambda_1 - \mu_1 \min(k_1, j_1)]$. Similarly for queue 2. Thus, a server is switched if that switch would help to balance the holding costs, after taking account of its effect on the two queues. The above policy will be referred to as the 'heuristic'.

The optimal, static and heuristic policies are compared by simulation. In order to model changes in demand, the simulation includes a sequence of alternating phases, with λ_1 and λ_2 changing values from one phase to the next. The performance measure in all cases is the total average holding cost, i.e. the simulation estimate of $E(c_1 j_1 + c_2 j_2)$. The following parameters are kept fixed: $\mu = 1$, $\xi = \eta = 0.1$, $c_1 = 1$, $c_2 = 2$, average phase duration = 100 (however, remember that parameters are renormalized to make the uniformization constant, Λ, equal to 1).

In figure 2.2, the average cost is plotted against the number of servers, N. λ_1 and λ_2 are in the ratio 1:100 during phase 1 and 100:1 during phase 2. Moreover, those arrival rates are increased with N so that the total offered load, $\rho_1 + \rho_2$, is equal to $3N/4$ (i.e., the system is reasonably heavily loaded).

The figure shows that the heuristic policy is almost as good as the optimal one, for all values of N. By contrast, the static policy (which is not entirely static; it changes the allocation within each phase, as the arrival rates change), is considerably more expensive and becomes worse with the increase in the number of servers.

A different comparison is illustrated in figure 2.3. Here the number of servers is fixed, $N = 4$, and the offered load increases, approaching saturation. The arrival rates are in the ratio 1:100 during phase 1 and 100:1 during

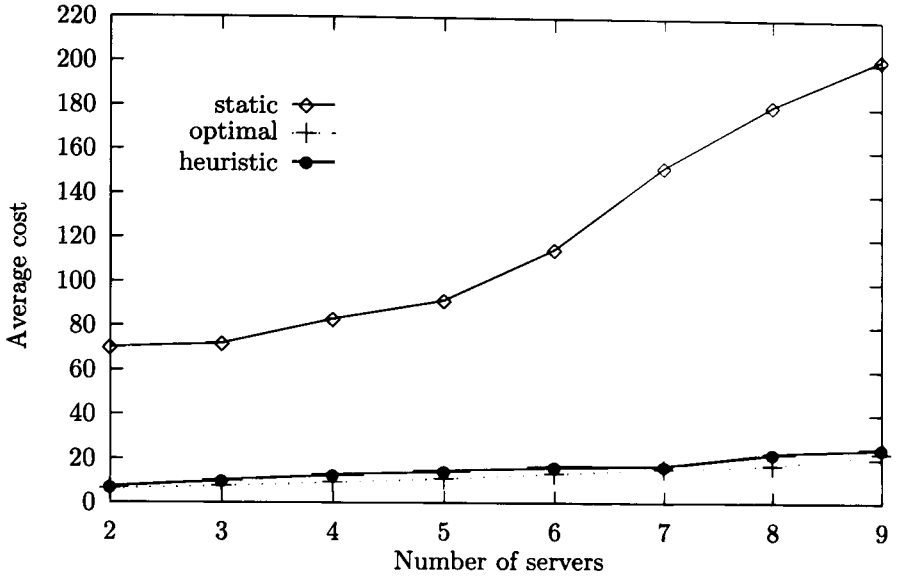

Figure 2.2: Policy comparisons: increasing N

phase 2, and are increased to produce the increase in total load.

This experiment shows even more emphatically that dynamic reconfiguration is advantageous. The cost of the static policy increases very fast, while the heuristic, which is again almost optimal, has much lower costs.

In these experiments, 200000 job completions were simulated, and approximately 1000 phase changes occurred. The longest simulation runs were for the optimal policy, because of the table look-ups.

2.6 Conclusions

This section has explored an initial problem of interest in the area of distributed processing and dynamic provision. The optimal reconfiguration policy for 2 server types has been computed and tabulated, subject to complexity

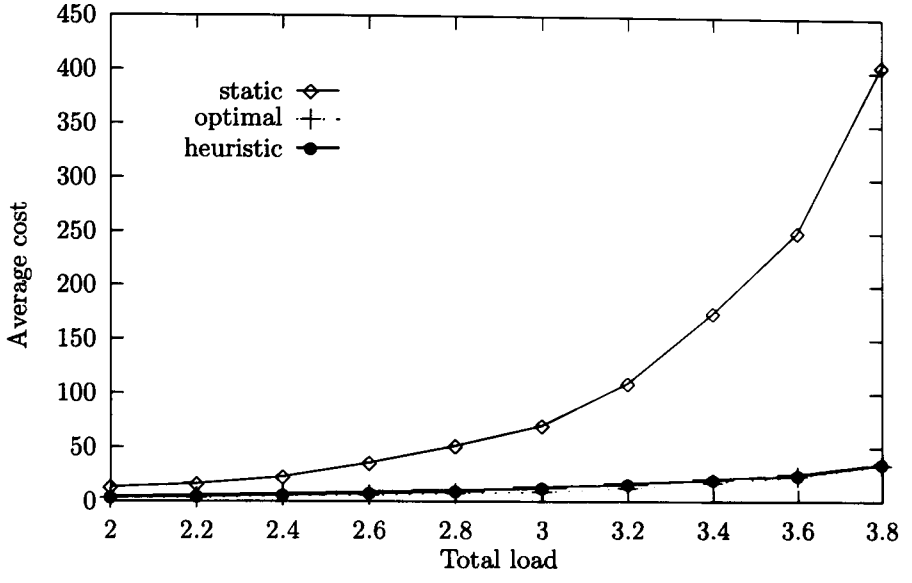

Figure 2.3: Policy comparisons: increasing loads

constraints imposed by the size of the state space and the ranges of parameter values. However, for practical purposes, an easily implementable heuristic policy is available. The encouraging results of figures 2.2 and 2.3 suggest that its performance compares quite favourably with that of the optimal policy.

A natural generalization of this problem would be to consider more than two job types and clusters. That would lead to a significantly more complex model, which is the object of the following section.

Chapter 3

Dynamic Server Allocation: M Job Types

3.1 Introduction

The model from the previous section is now generalized: consider a system consisting of a pool of N servers, split into M heterogeneous clusters of sizes k_1, k_2,...,k_M, where $\sum_{i=1}^{M} k_i = N$. Cluster i is dedicated to a queue of jobs of type i ($i = 1, ..., M$). Job types may for example include short web accesses or long database searches. Different types of job have different response time requirements (e.g., some may be less tolerant of delays than others). It is possible to reassign any server from one queue to another, but the process is generally not instantaneous and during it the server becomes unavailable. In those circumstances, a reconfiguration policy would specify, for any given parameter set (including costs), and current state, whether to switch a server or not.

3.2 The Model

The extended model is illustrated in Figure 3.1. Jobs of type i arrive according to an independent Poisson process with rate λ_i, and join a separate unbounded queue ($i = 1, 2, ..., M$). Their required service times are distributed exponentially with mean $1/\mu_i$. The cost of keeping a type i job in the system is c_i per unit time ($i = 1, 2, ..., M$). These 'holding' costs reflect the relative importance, or willingness to wait, of the M job types.

The Poisson arrivals assumption may be justified on the grounds that, in a Grid environment, many individual users submit jobs to the computational cluster. The arrival process of a given type is formed by merging a large number of independent sources, and is therefore approximately Poisson.

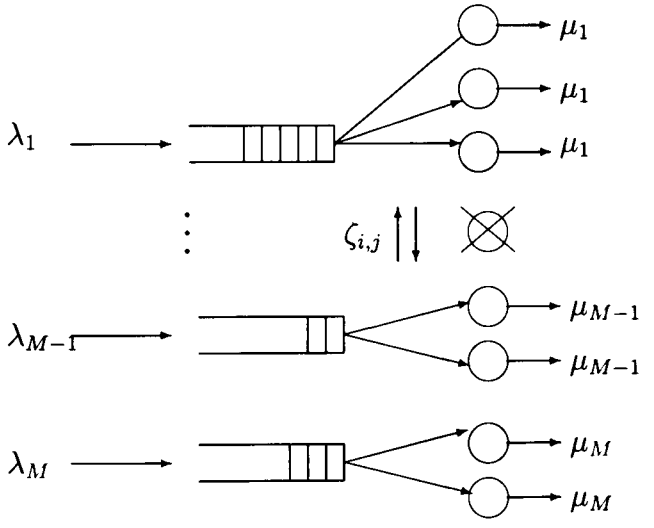

Figure 3.1: M reconfigurable heterogeneous clusters

The same reasoning applies from the model with only 2 servers. Any server currently allocated to queue i may be switched to queue j. Such a switch costs $c_{i,j}$ and takes an interval of time distributed exponentially with

mean $1/\zeta_{i,j}$, during which the server cannot serve jobs. It is assumed that switches are initiated at job arrival or departure instants. Indeed, it is at those instants that switches may become advantageous, and if they do, they should be performed without delay. Also, it is assumed that the switching policy employed is stationary, i.e., switching decisions may depend on the current state but not on past history.

Any job whose service is interrupted by a switch returns to the appropriate queue and resumes service from the point of interruption when a server becomes available for it.

The system state at any time is described by the triple, $S = (\mathbf{j}, \mathbf{k}, \mathbf{m})$ where $\mathbf{j} = (j_1, j_2, ..., j_M)$ is the vector of current queue sizes (j_i is the number of jobs in queue i, including those being served), $\mathbf{k} = (k_1, k_2, ..., k_M)$ is the vector of current server allocations (k_i servers allocated to queue i) and $\mathbf{m} = (m_{i,j})_{i,j=1}^{M}$ is the matrix of switches currently in progress ($m_{i,j}$ servers being switched from queue i to queue j, $m_{i,i} = 0$). The valid states satisfy $\sum_{i=1}^{M} k_i + \sum_{i,j=1}^{M} m_{i,j} = N$.

Under the above assumptions, the system is modelled by a continuous time Markov process. The transition rates of that process depend on the switching policy, i.e. on the decisions (actions) taken in various states. Denote by $r_d(S, S')$ the transition rate from state S to state S' ($S \neq S'$), given that action d is taken. The possible actions are (a) do nothing, or (b) initiate a switch from queue i to queue j (if $k_i > 0$ and $i \neq j$). These actions are represented by $d = 0$ (do nothing) and $d = 1, 2, ..., M(M-1)$.

The values of $r_d(S, S')$, for $S = (\mathbf{j}, \mathbf{k}, \mathbf{m})$, $S' = (\mathbf{j}', \mathbf{k}', \mathbf{m}')$ and $d = 0$, are given by the following (where $i, j = 1, ..., M$):

$$r_0(S,S') = \begin{cases} \lambda_i & \text{if } \mathbf{j}' = \mathbf{j} + \mathbf{e}_i \\ \min(j_i, k_i)\mu_i & \text{if } \mathbf{j}' = \mathbf{j} - \mathbf{e}_i \\ m_{i,j}\zeta_{i,j} & \text{if } \mathbf{m}' = \mathbf{m} - \mathbf{e}_{i,j} \text{ and } \mathbf{k}' = \mathbf{k} + \mathbf{e}_j \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{e}_i$ is the ith unit vector, and $\mathbf{e}_{i,j}$ is the matrix which has 1 in position (i,j) and zeros everywhere else.

The corresponding rates when $d \neq 0$ and the action taken is to switch a server from queue a to queue b ($a \neq b$) are obtained by replacing, in S', $\mathbf{k}'$ by $\mathbf{k}' - \mathbf{e}_a$ and $\mathbf{m}'$ by $\mathbf{m}' + \mathbf{e}_{a,b}$. Note that, in cases $d \neq 0$, there is a zero-time transition which changes k_a and $m_{a,b}$, and then an exponentially distributed interval with mean $1/r_d(S,S')$, after which the state jumps to S'.

The total transition rate out of state S, given that action d is taken, $r_d(S)$, is equal to:

$$r_d(S) = \sum_{S'} r_d(S,S') \; .$$

3.3 Computation of the Optimal Policy

Once again, for the purposes of optimization, it is convenient to apply the technique of uniformization to the Markov process, introducing 'fictitious' transitions which do not change the system state, so that the average interval between consecutive transitions ceases to depend on the state, and then embedding a discrete-time Markov chain at transition instants. First, find a constant, Λ, such that $r_d(S) \leq \Lambda$ for all S and d. A suitable value for Λ is

now

$$\Lambda = \sum_{i=1}^{M} \lambda_i + N\mu + N\zeta \ , \tag{3.1}$$

where $\mu = max(\mu_i)$ is the largest service rate and $\zeta = max(\zeta_{i,j})$ is the largest switching rate.

A Markov chain can now be constructed as described earlier in section 2.3. The finite-horizon optimization problem can be formulated in exactly the same way. To recap, $V_n(S)$ is the minimal expected total cost incurred during n consecutive steps of the Markov chain, given that the current system state is S. The complexity of the problem has increased now because the state space of the system is considerably increased. Any sequence of actions which achieves the minimal cost $V_n(S)$ constitutes an 'optimal policy' with respect to the initial state S, cost parameters, event horizon n, and discount factor α.

Suppose that the action taken in state S is d. This incurs an immediate cost of $c(d)$, equal to $c_{i,j}$ if the action taken is to switch a server from queue i to queue j. In addition, since the average interval between transitions is 1, each type i job in the system incurs a holding cost c_i. The next state will be S', with probability $q_d(S, S')$, and the minimal cost of the subsequent $n-1$ steps will be $\alpha V_{n-1}(S')$. Hence, the quantities $V_n(S)$ satisfy the following recurrence relations:

$$V_n(S) = \sum_{i=1}^{M} j_i c_i + \min_d \left[c(d) + \alpha \sum_{S'} q_d(S, S') V_{n-1}(S') \right] \ . \tag{3.2}$$

Thus, starting with the initial values $V_0(S) = 0$ for all S, one can compute $V_n(S)$ in n iterations. Once again, in order to make the state space finite, the

queue sizes are bounded at some level, $j_i < J$ $(i = 1, ... M)$. Then, if $V_{n-1}(S)$ has already been computed for some n and for all S, the complexity of computing $V_n(S)$, for a particular state S, is roughly constant. There are no more than $2M+M(M-1)$ states S' reachable from state S, and $M(M-1)+1$ actions to be compared (corresponding to the $M(M-1)$ possible switches from queue i to queue j and action $d = 0$ to do nothing). The best action to take in that state, and for that n, is indicated by the value of d that achieves the minimum in the right-hand side of (2.2). Since there are on the order of $O(J^M N^{M-1+M(M-1)})$ states altogether, the computational complexity of one iteration is on the order of $O(J^M N^{M-1+M(M-1)})$, and hence the overall complexity of solving (2.2) and determining the optimal switching policy over a finite event horizon of size n, is on the order of $O(nJ^M N^{M-1+M(M-1)})$.

Following the same reasoning from section 2.3, consider the infinite-horizon optimization when the discount factor α is strictly less than 1. When the optimal actions depend only on the current state, S, and not on n, the policy is said to be 'stationary'.

An argument similar to the one preceding (3.2) leads to the following equation for $V(S)$:

$$V(S) = \sum_{i=1}^{M} j_i c_i + \min_d \left[c(d) + \alpha \sum_{S'} q_d(S, S') V(S') \right] . \qquad (3.3)$$

The optimal policy (i.e. the best action in any given state) is specified by the value of d that achieves the minimum in the right-hand side of (3.3).

Equation (3.3) can be solved by applying the 'policy improvement' algorithm (see Dreyfus and Law [13]). This iterative algorithm can be applied to

the present optimization problem as follows.

Step 1. Start by making an initial guess about the optimal policy, i.e. construct an initial mapping, $d = f(S)$, from system states to action indices. This could be a simple heuristic such as the $c\mu$-rule (see [4]).

Step 2. Treat this guess as the optimal stationary policy, and compute the corresponding discounted costs, V^f, by solving the large set of simultaneous linear equations:

$$V^f(S) = \sum_{i=1}^{M} j_i c_i + \left[c(f(S)) + \alpha \sum_{S'} q_{f(S)}(S, S') V^f(S') \right] . \qquad (3.4)$$

Step 3. Now try to 'improve' policy f. For every state S, find the action $d^*(S)$ which achieves the minimum value in:

$$\sum_{i=1}^{M} j_i c_i + \min_d \left[c(d) + \alpha \sum_{S'} q_d(S, S') V^f(S') \right] . \qquad (3.5)$$

In other words, minimize the total cost in state S, assuming that *after the current operation*, policy f will be used.

Step 4. If action $d^*(S) = f(S)$ for all states S, then the policy f cannot be improved; it is optimal. Otherwise, the next guess for the optimal policy is $f(S) = d^*(S)$; repeat from step 2.

The computational complexity of this algorithm is determined by the complexity of each iteration, which is dominated by step 2, and by the number of iterations. The simultaneous equations can be represented in matrix and vector form as:

$$V = C + \alpha QF(V) \tag{3.6}$$

where V is the matrix of unknowns, C is the vector of holding and switching costs, Q is the matrix of transition probabilities from state S to state S', and $F(V)$ is an appropriate rearrangement of the elements of V.

An iterative method has been used to solve the set of simultaneous linear equations given in 3.6. Start with an initial approximation to V, such as the holding cost in the current state, $V_0(S) = \sum_{i=1}^{M} j_i c_i$, then at the n th iteration compute

$$V_n = C + \alpha QF(V_{n-1}) \tag{3.7}$$

Since Q is a stochastic matrix and $\alpha < 1$, this schema converges geometrically. This iterative solution is more efficient than Gaussian elimination for such a large state space, unless α is very close to 1.

3.4 Experimental results

First, consider the optimal switching decisions for the model with $N = 3$ and $M = 3$. In this example, the arrival and service parameters of the three job types are again the same, but waiting times for type 1 are twice as expensive as those for types 2 and 3. The discount factor is $\alpha = 0.95$. The stationary

optimal policy for states where $k_1 = k_2 = k_3 = 1$ and $j_1 = 0$ is shown in table 3.1. The truncation level used in the computation was $J = 20$, but the table stops at $j_2 = j_3 = 10$; the actions do not change beyond that level. Actions d are numbered as follows:

d=0, do nothing;

d=1, switch a server from queue 1 to queue 2;

d=2, switch a server from queue 2 to queue 1;

d=3, switch a server from queue 1 to queue 3;

d=4, switch a server from queue 3 to queue 1;

d=5, switch a server from queue 2 to queue 3;

d=6, switch a server from queue 3 to queue 2.

Again it is observed that switching is discouraged, compared to the $c\mu$-rule. For example, when $(j_1, j_2, j_3) = (0, 2, 1)$ the optimal decision is to do nothing, even though a job of type 2 is not being served whilst a server at queue 1 remains idle. Only when $(j_1, j_2, j_3) = (0, 3, 1)$ is the decision made to switch a server from queue 1 to queue 2.

The stationary optimal policy for states where $k_1 = 1, k_2 = 2, k_3 = 0$ and $j_1 = 0$ is shown in table 3.2 (all other parameters remain unchanged from those used in table 3.1. Similarly, we see switching discouraged when $(j_1, j_2, j_3) = (0, 0, 3)$ or $(j_1, j_2, j_3) = (0, 0, 4)$. The optimal decision is to do nothing, even though jobs of type 2 are once again not being served while a server at queue 1 remains idle.

Next, consider the optimal switching decisions for the model with $N = 4$ and $M = 3$. In this example, the arrival and service parameters of the three

		j_3										
		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	5	5	5	5	5	5	5	5	5
	1	0	0	0	3	3	3	3	3	3	3	3
	2	1	0	0	3	3	3	3	3	3	3	3
	3	1	1	1	1	3	3	3	3	3	3	3
	4	1	1	1	1	1	3	3	3	3	3	3
j_2	5	1	1	1	1	1	1	3	3	3	3	3
	6	1	1	1	1	1	1	1	3	3	3	3
	7	1	1	1	1	1	1	1	1	3	3	3
	8	1	1	1	1	1	1	1	1	1	3	3
	9	1	1	1	1	1	1	1	1	1	1	3
	10	1	1	1	1	1	1	1	1	1	1	1

Table 3.1: Optimal actions: non-zero switching times, $N = 3$, $M = 3$, $k_1 = k_2 = k_3 = 1$, $j_1 = 0$, $\lambda_1 = \lambda_2 = \lambda_3 = 0.111$, $\mu_1 = \mu_2 = \mu_3 = 0.111$, $c_1 = 2$, $c_2 = c_3 = 1$, $\zeta_{i,j} = 0.111 (i \neq j)$

		j_3										
		0	1	2	3	4	5	6	7	8	9	10
	0	5	5	5	5	5	5	5	5	5	5	5
	1	0	5	5	5	5	5	5	5	5	5	5
	2	0	3	3	3	3	3	3	3	3	3	3
	3	0	3	3	3	3	3	3	3	3	3	3
	4	0	3	3	3	3	3	3	3	3	3	3
j_2	5	1	1	3	3	3	3	3	3	3	3	3
	6	1	1	3	3	3	3	3	3	3	3	3
	7	1	1	1	3	3	3	3	3	3	3	3
	8	1	1	1	3	3	3	3	3	3	3	3
	9	1	1	1	1	3	3	3	3	3	3	3
	10	1	1	1	1	3	3	3	3	3	3	3

Table 3.2: Optimal actions: non-zero switching times, $N = 3$, $M = 3$, $k_1 = 1$, $k_2 = 2$, $k_3 = 0$, $j_1 = 0$, $\lambda_1 = \lambda_2 = \lambda_3 = 0.111$, $\mu_1 = \mu_2 = \mu_3 = 0.111$, $c_1 = 2$, $c_2 = c_3 = 1$, $\zeta_{i,j} = 0.111 (i \neq j)$

job types are again the same, and waiting times for type 1 are still twice as expensive as those for types 2 and 3. The discount factor is again $\alpha = 0.95$. The stationary optimal policy for states where $k_1 = 1$, $k_2 = 2$, $k_3 = 1$ and $j_1 = 1$ is shown in table 3.3. Switching is very discouraged, with most of the optimal decisions shown being to do nothing. This can be explained since there is a job of type 1 present which is currently being served. It is optimal to continue to serve this job, for the truncation levels of j_2 and j_3 calculated. However, the presence of non-zero switching times can also encourage switching compared to the $c\mu$-rule. When $(j_1, j_2, j_3) = (1, 0, 0)$, the optimal decision is to switch a server from queue 2 to queue 1, even though no unserved jobs of type 1 are waiting. This seems reasonable, since job type 1 is the most expensive to store. There are currently 2 idle servers at queue 2, so the decision is made to switch one of these over to queue 1, in case a job of type 1 arrives.

As the model has increased in complexity from 2 to M servers, characterizing the optimal policy explicitly in terms of the parameters is even less feasible. Instead, as in the previous section, one approach for using dynamic optimization in practice is to pre-compute the optimal policy for a wide range of parameter values, and store a collection of tables such as table 3.1. Then, having monitored the system and estimated its parameters, the optimal policy could be obtained by a table look-up. This is still feasible, but will now consume even more storage.

The third approach is again to formulate a heuristic policy which (a) is simply characterized in terms of the parameters, and (b) performs acceptably well, compared with the optimal policy. That is what is proposed in the

		j_3										
		0	1	2	3	4	5	6	7	8	9	10
	0	2	2	5	5	5	5	5	5	5	5	5
	1	0	0	0	5	5	5	5	5	5	5	5
	2	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	0	0	0	0
j_2	5	0	0	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	0	0

Table 3.3: Optimal actions: non-zero switching times, $N = 4$, $M = 3$, $k_1 = 1$, $k_2 = 2$, $k_3 = 1$, $j_1 = 1$, $\lambda_1 = \lambda_2 = \lambda_3 = 0.0909$, $\mu_1 = \mu_2 = \mu_3 = 0.0909$, $c_1 = 2$, $c_2 = c_3 = 1$, $\zeta_{i,j} = 0.0909 (i \neq j)$

following section.

3.5 Heuristic Policies

The heuristic from section 2.5 is now generalized. When the number of queues does not exceed the number of servers, it is possible to do no switching at all. Allocate the servers roughly in proportion to the offered load, $\rho_i = \lambda_i/\mu_i$, and to the holding cost, c_i, for each type. In other words, set

$$k_i = \left\lfloor N \frac{\rho_i c_i}{\sum_{j=1}^{M} \rho_j c_j} + 0.5 \right\rfloor \; (i = 1, ..., M-1) \; ; \; k_M = N - \sum_{j=1}^{M-1} k_j \; ,$$

if all k_i are non-zero. If any k_i is zero, replace k_i with 1 and the largest k_j $(i \neq j)$ by $k_j - 1$. Repeat this process until all k_i are non-zero. Having made the allocation, leave it fixed as long as the offered loads and costs remain

the same. This will be referred to as the 'static' policy. It certainly has the virtue of simplicity, and also provides a comparator by which the benefits of dynamic reconfiguration can be measured.

Once again, the idea behind this dynamic heuristic policy is to attempt to balance the total holding costs of the different job types. That is, the policy tries to prevent the quantities $j_i c_i$ $(i = 1, ..., M)$ from diverging. The following rule is applied:

1. Calculate the following for each of the $M(M-1)$ possible switches from queue a to queue b ($a \neq b$ and $k_a > 0$):

$$c_b \left\{ j_b + \frac{1}{\zeta_{a,b}} [\lambda_b - \mu_b \min(k_b, j_b)] \right\}$$

$$- K c_a \left\{ j_a + \frac{1}{\zeta_{a,b}} [\lambda_a - \mu_a \min(k_a - 1, j_a)] \right\},$$

   where K is a constant used to discourage too many switches from being initiated. The best value of K depends on the total load. For heavily loaded systems, $K = 5$ has been used.

2. Find the maximum of all quantities calculated in 1; if it is strictly positive, this will be the most advantageous switch to initiate. Take the action $d \neq 0$ corresponding to this switch. Otherwise, take action $d = 0$.

This rule is once again based on approximating the effects of a switch. If j_b jobs of type b are present and k_b servers are available for them, then the average queue b increment during an interval of length x may be estimated as $x[\lambda_b - \mu_b \min(k_b, j_b)]$. Similarly for queue a, except that if a server is switched

from queue a then the available servers for this queue drops to $k_a - 1$. Thus, a server is switched if that switch would help to balance the holding costs, after taking account of its effect on the M queues. The above policy will be referred to as the 'heuristic'.

The optimal, static and heuristic policies are compared by simulation. The performance measure in all cases is the total average holding cost, i.e. the simulation estimate of $E(\sum_{i=1}^{M} c_i j_i)$. In all experiments, the parameters given below are renormalized to make the uniformization constant, Λ, equal to 1.

In figure 3.2, the average cost is plotted against the number of servers, N when $M = 3$. The following parameters are used: $1000\lambda_1 = \lambda_2 = \lambda_3$, $b_1 = 1000b_2 = 1000b_3$, $c_1 = 2$, $c_2 = 1$, $c_3 = 1$. Switching rates are equal and given by $\zeta_{i,j} = \mu_2/10 = \mu_3/10$. Arrival rates are increased with N so that the total offered load, $\rho_1 + \rho_2 + \rho_3$, is equal to $4N/5$ (i.e., the system is heavily loaded). This models a system where type 1 jobs are long and types 2 and 3 are much shorter. Requests of type 1 arrive at a much slower rate than for types 2 and 3, although the total load for each job type is the same.

Another comparison when $M = 3$ is shown in figure 3.3. Here, the number of servers is fixed at $N = 4$ and the total load increases. All arrival rates are equal. The following parameters are used: $\mu_1 = \mu_2 = \mu_3 = 1$, $\zeta_{i,j} = 0.1$, $c_1 = 2$, $c_2 = 1$, $c_3 = 1$.

Figures 3.2 and 3.3 demonstrate clearly the benefit of dynamic reconfiguration of servers when the number of job types is increased to $M = 3$. The static policy performs poorly as the number of servers or the load is increased, while the heuristic policy performs almost as well as the optimal

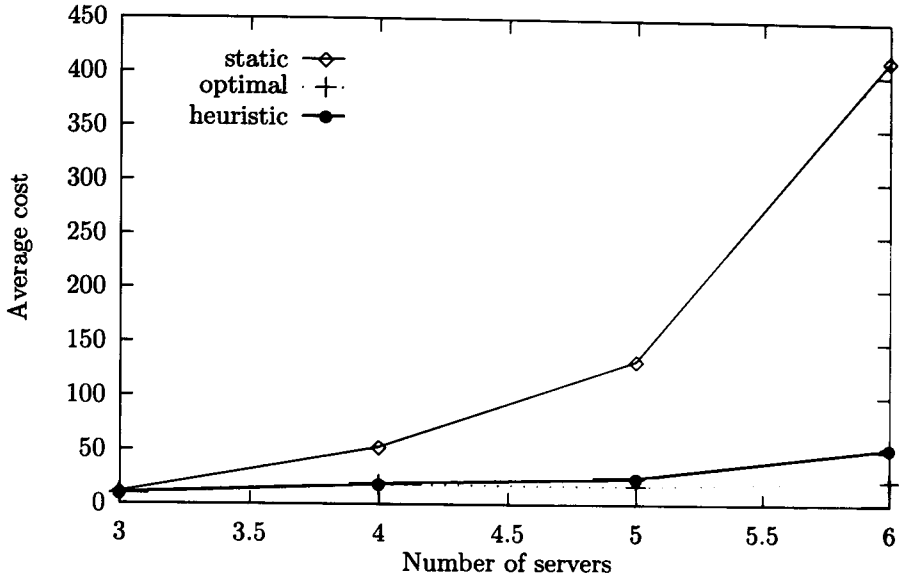

Figure 3.2: Policy comparisons: $M = 3$ and increasing N

policy. In each case, choosing dynamic reconfiguration dramatically reduces the average holding costs.

In all of these experiments, 200000 job completions were simulated. Where phase changes were simulated, approximately 1000 phase changes occurred during the duration of the simulation. The longest simulation runs were for the optimal policy, because of the table look-ups.

Calculations of the table look-ups for the optimal policy have been executed on a Linux machine with an Intel Xeon 2.80GHz processor and 1GB RAM. To calculate the optimal policy for $N = 4$, $M = 3$ and a truncated queue size of $J = 30$ requires 223MB of available memory. As an illustration of the complexity of the calculations and the large size of the state space, if the number of servers is increased to $N = 6$, with $M = 3$ and $J = 30$, calculating the optimal policy now requires 1.564GB of available memory.

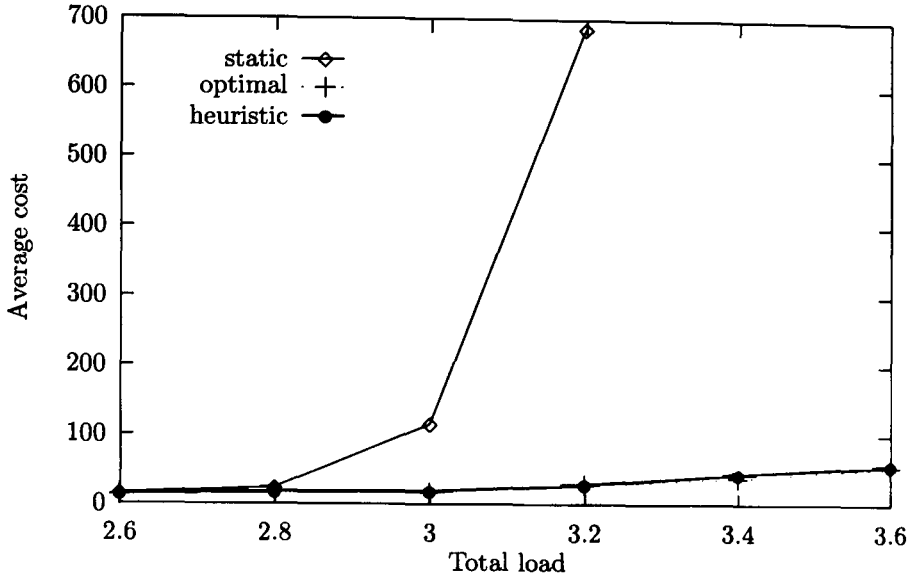

Figure 3.3: Policy comparisons: $M = 3$ and increasing loads

Each table of decisions for $N = 4$, $M = 3$ and $J = 30$, when calculated using the Policy Improvement described in Section 3.3, took approximately 16 minutes. The Policy Improvement algorithm took 10 iterations to converge to the optimal policy, with the initial 'guess' of the policy set to the heuristic policy. Solving the large set of simultaneous equations using an iterative process is the most computationally expensive stage of the Policy Improvement algorithm. Initializing the cost matrix to the holding cost of the current state, this set of equations takes approximately 180 iterations to converge to within an accuracy of 0.01 in the first Policy Improvement iteration. This reduces upon each iteration. The Policy Improvement algorithm converges to the optimal policy within approximately 6 iterations.

3.6 Conclusions

The problem of interest examined in the previous chapter 2 has been generalized to consider more than two job types and clusters. In this section, the optimal reconfiguration policy for a general number of server types, M, has been computed and tabulated, subject to complexity constraints imposed by the size of the state space and the ranges of parameter values. The heuristic from section 2.5 has been extended. This is easily implementable and practical to use. Once again, encouraging results suggest that its performance compares quite well with that of the optimal policy. These results are presented in figures 3.2 and 3.3.

The experiments reported here were carried out for systems with relatively small numbers of servers and job types. The main constraint is the complexity of computing the optimal policy, which is necessary as a yardstick of comparison with the heuristics. The latter can be implemented and/or simulated easily in much larger systems.

Chapter 4

Optimal Tree Structures

4.1 Introduction

The context for this chapter is again a large service provisioning system such as the *Computing Grid*: a network consisting of large clusters of resources (servers) offering services to a large community of users. Incoming service requests (jobs) do not have to be executed on a particular server but may be sent anywhere; their destination is transparent to the user.

Within such provision of services, it is important that the clusters and their controllers are configured in an efficient manner. In the previous chapter the dynamic allocation of resources to incoming job types was discussed. Now for consideration is the underlying structure of these compute resources, regardless of the type of service or job offered.

Conceptually, the nodes in the network are divided into *masters*, who make routing decisions, and *servers*, who execute jobs. In practice, a master and a server may be co-located on one processor; in that case some of

the latter's processing capacity is used for purposes of control and some for serving jobs.

A simple and convenient network structure that accommodates these different types of nodes is the tree. The non-leaf nodes are masters who distribute jobs among nodes under their control; the leaf nodes are servers. Other, more complex organizations are possible, but attention shall be restricted to trees for reasons of clarity and tractability.

There are performance trade-offs between different tree configurations. They arise from the fact that the average processing time of a job at a master node is roughly proportional to the number of nodes under the latter's immediate control. Those nodes (they may be either masters or servers), will be referred to as the master's *dependents*. That proportionality is due to the nature of the processing carried out by a master: it must check the state of its dependents (an example of a query mechanism can be found in the Condor system [44], [28]). Hence, the 'flatter' the tree, the more dependents a master node has, the longer its average processing times. Nodes with many dependents can easily become bottlenecks when demand is high; their queues grow long, leading to large response times.

The bottleneck problem can be alleviated or avoided by making the tree 'taller', introducing more master nodes, with fewer dependents per master. However, the consequent speeding up of processing times must then be set against an increase in the number of times a job has to be processed, plus additional transfer delays.

The evaluation of the above trade-offs is the subject of this chapter. The 'dual' problem will be focussed upon: given the overall rate of demand, how

should the network be configured in order to optimize its performance? This problem does not appear to have been studied before. It is interesting because recent developments in Grid technology have made it possible to reconfigure the underlying tree structure dynamically. To simplify the analysis, it shall be assumed that the necessity for reconfiguration occurs rarely. That is, the load remains reasonably constant for a sufficiently long period to allow the system to reach steady-state. Also, since reconfigurations are rare, their cost will be ignored.

The performance measure used as an optimization criterion is the average response time, i.e. the interval between the arrival of a job into the system and the completion of its service. That interval may include waiting and processing at one or more master nodes, transfer delays between nodes, and waiting and processing at a server node. Different policies for distributing jobs among the nodes in the tree (both master and server) will be examined. For some of them, the analytical evaluation of average response times is approximate.

4.2 Two-Level Tree Structures

4.2.1 The Model

The preliminary model considers i levels of master nodes ($i \in (1, 2)$). A *flat* structure where $i = 1$ is illustrated in figure 4.1. It is assumed that the single master node on level 1 will take a longer time to make a routing decision the greater the number of dependent service nodes beneath it. Indeed, the time

for this decision to be made has been modelled as proportional to the number of dependent nodes. Hence, this flat structure may often not be optimal: the master node may become over-loaded if this decision making process is too slow. Introducing a further tier of nodes helps to relieve this problem: now each master node has fewer dependents between which to choose. A tree structure to be optimized in this section with $i = 2$ is illustrated in figure 4.2.

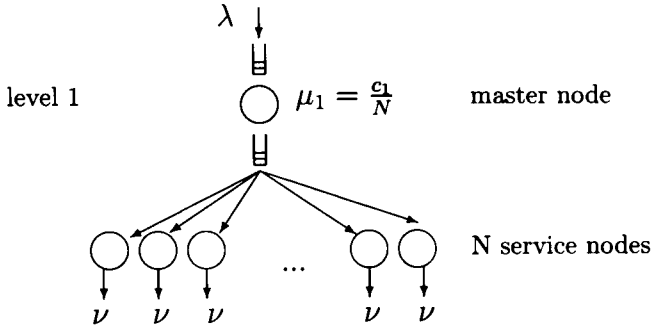

Figure 4.1: Flat structure

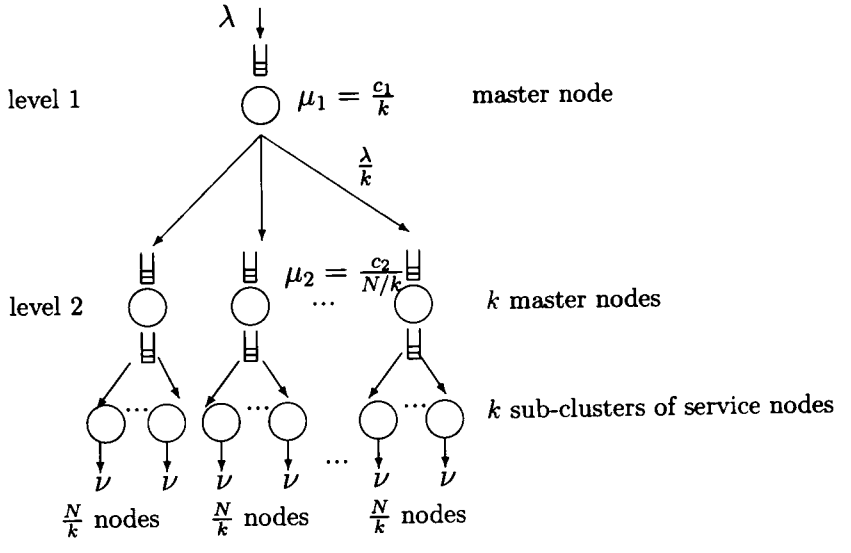

Figure 4.2: Service cluster split into k sub-clusters

Jobs arrive according to an independent Poisson process with rate λ. Their required service time is distributed exponentially with mean $1/\nu$. The average service rate μ_i of a master at a given node of the tree at level i is inversely proportional to the number of dependents of that node, i.e.

$$\mu_i = \frac{c_i}{n} , \tag{4.1}$$

where c_i is a constant of proportionality at level i and n is the number of dependents. It is also assumed that these constants of proportionality may be different at different levels of the tree. This is particularly likely for a large scale Computing Grid, across which the querying of geographically distant nodes at the master level will take more time on average than a cluster manager querying its local service nodes. In addition, independent transfer delays T_i are assumed when sending jobs from one node to another. Jobs arrive into a FIFO queue in front of each master node. Following a routing decision, the master node distributes jobs among its dependents. Different policies for distributing jobs among the nodes in the tree may be considered. These include

1. Each dependent has a separate queue; the master places new jobs into those queues in random order.

2. Each dependent has a separate queue; the master places a new job into the queue which is currently shortest.

3. Each dependent has a separate queue; the master places new jobs into those queues in cyclic order.

4. Dependents at a final service cluster level have a joint queue

Simulation results have shown that, at the service cluster level, n parallel separate queues using policies 2 or 3 may be approximated by a single $M/M/n$ queue (described further in section 4.3.4) so long as n is reasonably large. This assumption is appropriate for any realistic Grid hosting environment. The model will continue to be formulated using an $M/M/n$ queue at each final sub-cluster. Each master node will perform a query to determine where a job should be routed, the service rate of which is inversely proportional to the number of its dependents. If a master node is also head of a final sub-cluster, the job service is modelled by an $M/M/n$ queue with service rate ν.

How the routing decision is made will affect average response times, particularly for a heavily loaded system. Simulations have been used to compare pure *random* choice with that of querying the shortest queue or using a cyclic order. The latter two methods of decision making give very similar average response times, both performing significantly better than random choice. In this preliminary model, detailed analysis has been performed for a model using only random choice. The other methods are described in more detail in section 4.3.

Any server may be assigned as either a master node or a service node. Known theoretical results [22] for average response times have been used to compare different configurations of the tree structure shown in figure 4.2. The service nodes are split into k different sub-clusters, each with its own master node responsible for routing an incoming job to an appropriate service

node in its cluster. Of interest is the particular value of k, which for a given number of servers N, arrival rate λ, service rate ν, transfer times T_i and constants of proportionality c_i, minimizes the overall average response time of the system.

4.2.2 Computation of The Optimal Tree Structure

The average response time may be evaluated as follows. The master node at level i querying its dependents is an $M/M/1$ system, with service rate μ_i inversely proportional to the number of dependents. The average sojourn time (sojourn times include waiting and processing at a particular level of the tree) for an $M/M/1$ system with offered load ρ_i (which is in this case equal to λ/μ_i, $i = 1, 2$) is given by

$$W_i = \frac{1}{\mu_i(1-\rho_i)} . \tag{4.2}$$

Each final service cluster of n nodes is modelled as an $M/M/n$ system. The average sojourn time for an $M/M/n$ system with arrival rate λ and offered load ρ (which now is equal to λ/ν) is given by

$$W_{final} = \frac{1}{\lambda}\left[\sum_{j=1}^{n-1}\frac{\rho^j}{(j-1)!} + \frac{\rho^n(n^2-n\rho+\rho)}{(n-1)!(n-\rho)^2}\right]\left[\sum_{j=0}^{n-1}\frac{\rho^j}{j!} + \frac{\rho^n}{(n-1)!(n-\rho)}\right]^{-1} . \tag{4.3}$$

A flat structure, as shown in figure 4.1, is possible so long as the master node is not saturated, i.e. $\rho_1 < 1$ where $\rho_1 = \lambda/\mu_1 = \lambda N/c_1$. So, for a non-saturated flat structure $c_1 > \lambda N$ is required. When this condition holds,

the average response time for the flat structure is given by

$$W = W_1 + W_{final} \,. \tag{4.4}$$

If the condition $c_1 > \lambda N$ does not hold, the master node in the flat structure will be over-loaded. It is now necessary to introduce a second network layer of k master nodes, as shown in figure 4.2, thereby reducing the service time for a master node to query its subset of dependents. The average response time will now be given by

$$W = W_1 + T_1 + W_2 + W_{final} \,. \tag{4.5}$$

The objective is to minimise the right-hand side of (4.5) with respect to k. The optimal value of k is difficult to obtain in closed form, but is easily and quickly computed numerically. At a service cluster, the arrival rate is λ/k and the available service rate is $N\nu/k$, so the offered load is $\lambda/N\nu$ regardless of k. If the overall system is not to be saturated, first required is $\lambda/N\nu < 1$. Then, for a given parameter set, k has upper and lower bounds so that no master node within the network becomes saturated. For the tree structure to be possible, $\rho_i < 1$ is required at each master node. At the first master node this gives $k < c_1/\lambda$. At the next master node this gives $k > \sqrt{\lambda N/c_2}$. Hence, possible values of k are within the range

$$\sqrt{\frac{\lambda N}{c_2}} < k < \frac{c_1}{\lambda} \,. \tag{4.6}$$

Average response times for each value of k within this range are evaluated

and compared to find the minimum response time, and hence the optimal value for k. This gives the optimal network configuration with a single layer of master nodes. As λ increases, the range of possible values for k given in (4.6) diminishes and eventually there will be no acceptable value of k. The system will be saturated and will not cope with the demand. At this point, another network layer could be introduced, and this is a topic discussed in section 4.3.

4.2.3 A simple heuristic

Although evaluating and comparing all feasible configurations of the network is not difficult, it may be a time-consuming task when the number of feasible possibilities is large. It is therefore useful to offer a simple heuristic configuration rule which, although possibly sub-optimal, provides acceptable solutions.

Consider the total offered load, $f(k)$, at the level 1 master node and one of the level 2 master nodes. It is given by

$$f(k) = \frac{\lambda k}{c_1} + \frac{\lambda N}{c_2 k^2} + \frac{\lambda}{N\nu} . \tag{4.7}$$

This total load may be minimized with respect to k to find an initial value for k given the number of service nodes N and the constants of proportionality c_1 and c_2, which represent the speed at which routing decisions may be made at different network levels. Differentiation of (4.7) and equating with zero to

determine a minimum leads to

$$k = \sqrt[3]{\frac{2Nc_1}{c_2}}\,. \tag{4.8}$$

This value for k matches the numerically obtained optimal value for k quite closely when the overall network load is low, and may be used as a starting configuration of the network. The numerical calculation to obtain the optimal value for k described in the previous section is relatively simple and can be used for dynamic network configuration.

4.2.4 Results

Graphs of average response time as k varies clearly show the benefit of carefully choosing the configuration of the tree. Figure 4.3 shows the effect of varying the number of master nodes in a system with $N = 100$, $c_1 = c_2 = 100$, $T_1 = 0.001$, $\lambda = 8$ and $\nu = 0.1$. This system is reasonably heavily loaded, with an offered load of 0.8. A 'flat' structure is not feasible, since $c_1 < \lambda N$. Introducing a second level of master nodes, the possible values for k are $3 \leq k \leq 12$, according to (4.6). The optimal value for k which minimises the average response time is $k = 4$. It performs considerably better than the 'poor' configurations with $k = 3$ or $k = 12$. The heuristic (4.8) suggests $k = 6$; its performance is only slightly worse than the optimum.

Figures 4.4 and 4.5 compare the optimal value of k with that predicted by the simple heuristic given in (4.7). In figure 4.4, the load is fixed at 0.8. The parameters are $\nu = 0.1$, $T_1 = 0.01$, $c_1 = c_2 = 100$. The number of servers N increases, with the arrival rate λ increasing in proportion in order

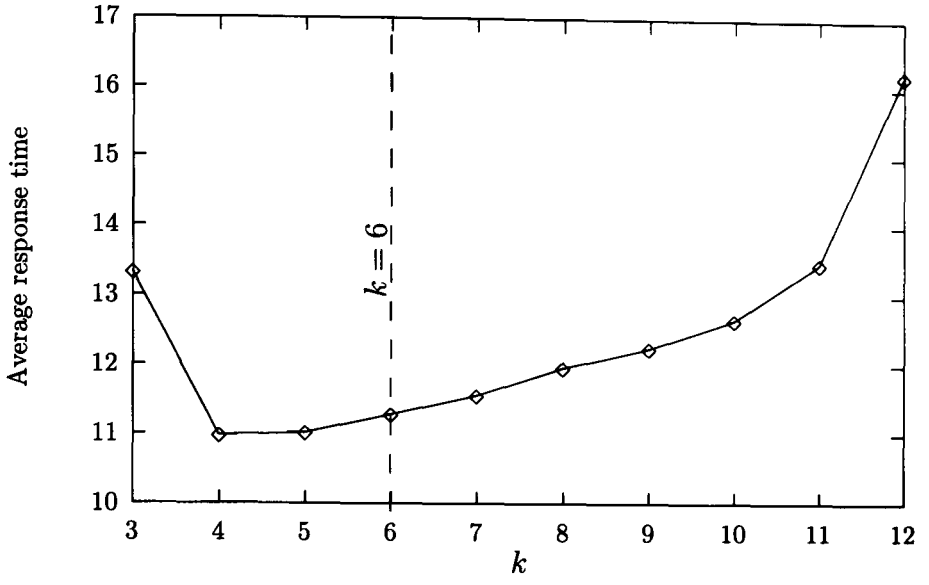

Figure 4.3: Performance of different configurations
$N = 200$, $c_1 = c_2 = 100$, $T_1 = 0.001$, $\lambda = 8$, $\nu = 0.1$, load= 0.8

to keep the load constant. The heuristic is observed to predict a consistently higher value for k than the optimum at this load. However, the difference in average response times between the heuristic and the optimal configurations is minimal.

In figure 4.5, the number of servers is fixed at $N = 100$, and the load increases with λ. Here $c_1 = c_2 = 100$, while the other parameters are the same as before. The heuristic yields $k = 5$, which is observed to under-estimate the optimum k when the load is low, and over-estimate it when the load is high. Again, the difference in performance between the heuristic and the optimal configurations is very small.

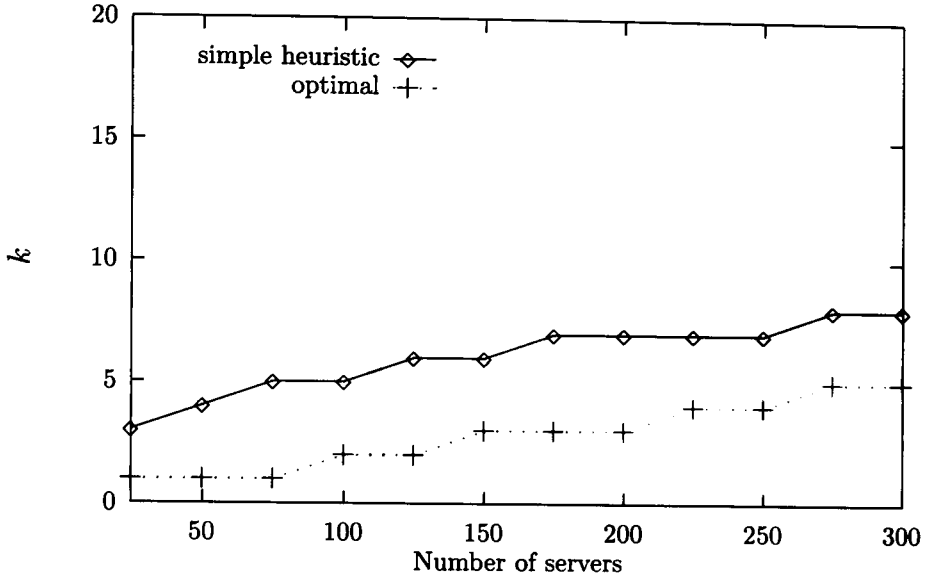

Figure 4.4: Optimal and heuristic configurations
$c_1 = c_2 = 100$, $T_1 = 0.001$, $\nu = 0.1$, load=0.8

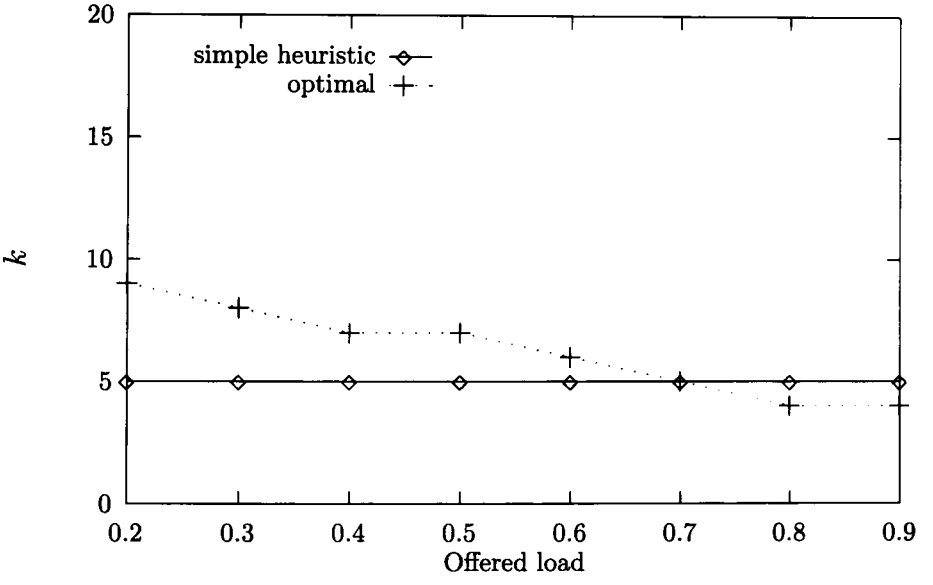

Figure 4.5: Optimal and heuristic configurations
$N = 100$, $c_1 = c_2 = 100$, $T_1 = 0.001$, $\nu = 0.1$

4.2.5 Conclusions

Through numerical analysis, a tree structure with a maximum of two levels has been decided upon which minimizes the overall average response time of the service provisioning system. The encouraging results of figures 4.4 and 4.5 suggest that the benefits of dynamic reconfiguration of the network as load changes will reduce long term average response times and help make such a hosting environment commercially viable. A simple heuristic is available for the configuration of a network with a given number of service nodes and specified decision making speeds. This heuristic gives an adequate configuration which may be improved upon as the network load increases.

As network load increases further, a further tier of master nodes will be necessary to avoid saturation. This is a problem of considerably increased complexity, and is discussed in section 4.3.

The model assumption that the routing decision made by a master node is one of random choice is not particularly realistic. Different models need to be considered in more detail, and their effect on overall average response times noted. It is more likely that a master node would make an *intelligent* decision based on shortest queue or round-robin. Theoretical results are available for the round-robin method of routing decision. This is also discussed in the following section 4.3.

4.3 M-Level Tree Structures

4.3.1 The Model

The previous model from section 4.2.1 is extended to include further tiers of master nodes. The network again consists of a number of master nodes, arranged in a tree with m levels, and a total of N server nodes. At one extreme, when $m = 1$, a single master node controls all N servers, illustrated previously in figure 4.1. At the other extreme, the master nodes form a binary tree, each controlling either 2 master nodes, or 2 or 3 servers (there is no point having a node with just 1 dependent).

Jobs arrive into the master node at the root of the tree according to a Poisson process with rate λ. Whether arriving at the root, or at a master node at level i in the tree, jobs join an unbounded queue and are processed in FIFO order. The processing times for masters at level i who have n_i dependents are distributed exponentially with mean $1/\mu_i(n_i)$, where

$$\mu_i(n_i) = \frac{c_i}{n_i} . \tag{4.9}$$

Here c_i is a constant of proportionality which may depend on the level. The constant c_i reflects the difficulty of querying dependents at level i; also, if a master node is physically co-located on a processor that is used to serve jobs, it takes into account the fraction of processing capacity available for purposes of control.

If the dependents of a master node are other masters, jobs are routed to one of them; the transfer from level i to level $i + 1$ is assumed to be an

independent random variable with mean T_i $(i = 1, 2, \ldots, m-1)$. That is, the average transfer time does not depend on the number of jobs which are in transit.

On the other hand, if the dependents of a master form a (presumably local) sub-cluster of server nodes, they share a common queue. Processing times at server nodes are distributed exponentially with mean $1/\nu$.

An example of a tree structure with three levels of master nodes ($m = 3$) is illustrated in figure 4.6.

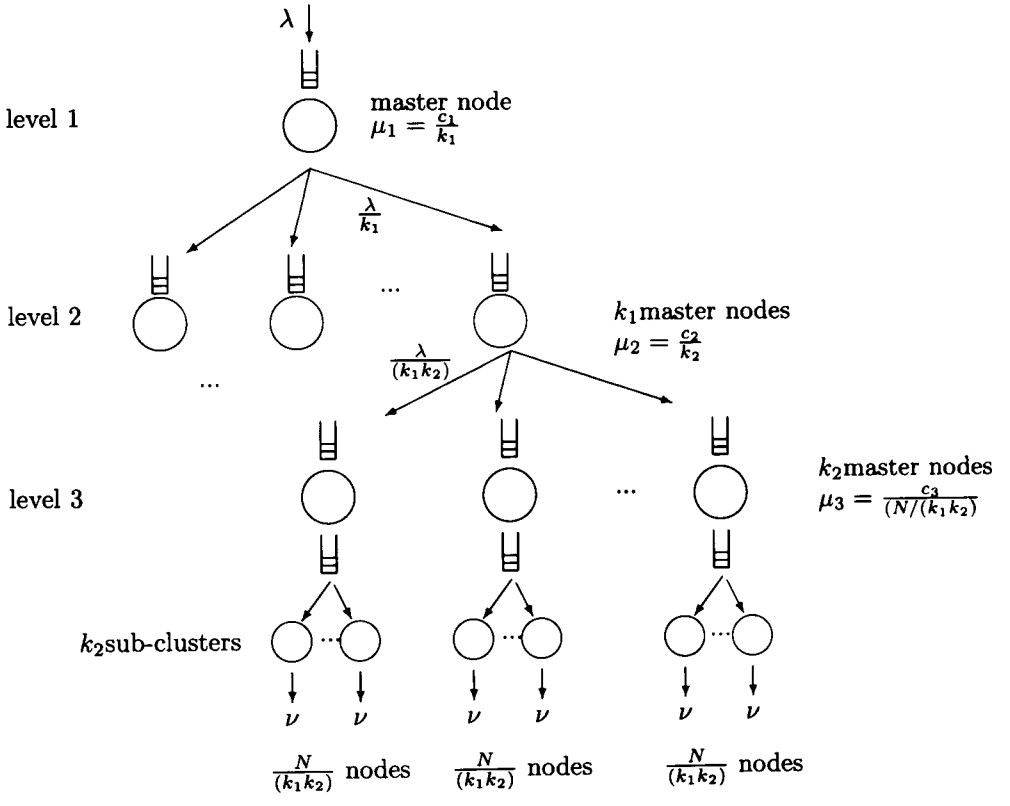

Figure 4.6: A tree with three levels of master nodes

The total average response time, W, in a network with m levels of master

nodes is given by

$$W = \sum_{i=1}^{m} W_i + \sum_{i=1}^{m-1} T_i + S_m \ , \tag{4.10}$$

where W_i, $i = 1, 2, \ldots, m$, is the average sojourn time at a master node at level i, T_i is the average transfer time from level i to level $i+1$. and S_m is the average sojourn time at a service sub-cluster at level m (sojourn times include waiting and processing).

Master nodes may use different routing policies to distribute jobs among dependent master nodes. Three such policies are now considered.

1. Uniform random routing: each dependent master is equally likely to be chosen as a destination for a job, regardless of previous decisions.

2. Shortest queue routing: send the job to the master whose queue is currently shortest.

3. Cyclic routing: if there are n dependent masters and the last job was sent to master j, send the next job to master $(j+1)$ *mod* n.

All three policies aim to balance the load; the arrival rate at a dependent master is $1/n$th of the arrival rate at the parent (if there are n dependents). However, the traffic processes are different, and those differences have performance implications.

Under policy 1, if the arrival process at the parent master is Poisson, then so is the arrival process at each of the dependents. Hence, all master nodes behave like M/M/1 queues, and a service sub-cluster containing n servers behaves like an M/M/n queue. That is the only model where the

total average response time in a network with $m > 1$ can be determined exactly.

Policy 2 is more efficient, but also more difficult to implement, since it requires queue size information about dependents. Moreover, it is very difficult to analyze. Although there are some numerical algorithms that may be applicable when the number of dependents is small (e.g., see [3]), simulation appears to be the only feasible evaluation tool in a general network setting.

Policy 3 is a good compromise in terms of both performance and analytical tractability. Simulation results (section 4.3.4) show that, for reasonable loads, cyclic routing is almost as efficient as the shortest queue policy. Average response times under cyclic routing can be estimated quite simply, by making a single approximating assumption: that the departure process from the parent node is Poisson (section 4.3.2). For those reasons, most of the numerical results in section 4.3.4 are obtained for networks with cyclic routing at master nodes.

4.3.2 Computation of the Optimal Tree Structure

Consider a master node at level i, with n_i dependents and n_{i-1} siblings, including itself (i.e., its parent node has n_{i-1} dependents; if $i = 1$, then $n_{i-1} = 1$). Assume that the departure process from the parent node is Poisson (this may or may not be an approximation).

If the job distribution policy at the parent node is uniform random routing, then this node behaves like an M/M/1 queue with service rate $\mu_i(n_i)$,

given by (4.9), and arrival rate λ_i, obtained from $\lambda_i = \lambda_{i-1}/n_{i-1}$ (with $\lambda_1 = \lambda$). The average sojourn time, W_i, is equal to

$$W_i = \frac{1}{\mu_i(n_i)(1-\rho_i)}, \tag{4.11}$$

where $\rho_i = \lambda_i/\mu_i(n_i)$ is the offered load.

The other case where a simple solution exists for W_i is when the parent node employs the cyclic routing policy. Then there are exactly $n_{i-1} - 1$ departures from the parent node between consecutive arrivals at this node. In other words, the interarrival interval is the sum of n_{i-1} i.i.d. random variables distributed exponentially with parameter λ_{i-1}. Hence, this node behaves like a GI/M/1 queue with an Erlang interarrival distribution (parameters λ_{i-1} and n_{i-1}), and service rate $\mu_i(n_i)$. The solution is (see [22])

$$W_i = \frac{1}{\mu_i(n_i)(1-\sigma_i)}, \tag{4.12}$$

where σ_i is the unique root of

$$\sigma_i = \left(\frac{\lambda_{i-1}}{\lambda_{i-1} + \mu_i(n_i) - \mu_i(n_i)\sigma_i}\right)^{n_{i-1}}, \tag{4.13}$$

in the range $0 < \sigma_i < 1$. This equation is solved iteratively.

A service sub-cluster whose master node is at level m, and which consists of n servers, is modelled as an $M/M/n$ queue with arrival rate λ_m and service rate ν. The offered load is equal to $\rho = \lambda_m/\nu$. The average sojourn time,

S_m, is given by

$$S_m = \frac{1}{\lambda_m}\left[\sum_{j=1}^{n-1}\frac{\rho^j}{(j-1)!} + \frac{\rho^n(n^2-n\rho+\rho)}{(n-1)!(n-\rho)^2}\right]p_0 \,. \tag{4.14}$$

where

$$p_0 = \left[\sum_{j=0}^{n-1}\frac{\rho^j}{j!} + \frac{\rho^n}{(n-1)!(n-\rho)}\right]^{-1} \,.$$

Since the external arrival rate into the network is λ, and there is a total of N service nodes available, each with service rate ν, the condition $\lambda < N\nu$ is necessary for stability. However, it is not sufficient. Each master node and each service sub-cluster must be stable, and those conditions depend on the structure of the tree. For example, if a single master node controls all N servers, i.e. $m = 1$, then, in addition to the condition $\lambda < N\nu$, the following must all hold: $\rho_1 = \lambda/\mu_1 = \lambda N/c_1 < 1$, or $\lambda < c_1/N$. When both inequalities are satisfied, the average response time is equal to

$$W = W_1 + S_1 \,, \tag{4.15}$$

with W_1 and S_1 given by (4.11) and (4.14), respectively.

If the condition $\lambda < c_1/N$ does not hold, a single master node will be overloaded, but a two-level tree may be stable. Introduce a second network layer of k_1 master nodes, assigning N/k_1 servers to each master (if k_1 is not a factor of N, make the sub-clusters as equal as possible). The master at level 1 is now stable if $\lambda < c_1/k_1$, while those at level 2 (assuming that the load is split equally) are stable if $\lambda/k_1 < c_2k_1/N$. Thus, the feasible 2-level

network configurations are the ones that satisfy

$$\sqrt{\frac{\lambda N}{c_2}} < k_1 < \frac{c_1}{\lambda} . \tag{4.16}$$

Within that range, some configurations are better than others. The average response time is given by

$$W = W_1 + T_1 + W_2 + S_2 , \tag{4.17}$$

where W_1, W_2, and S_2 are obtained from (4.11), (4.12) (assuming cyclic routing), and(4.14), respectively, with $n_1 = k_1$ and $n_2 = N/k_1$. This metric can be minimized with respect to k_1. The optimal value of k_1 is easily and quickly computed numerically. Some results are shown in section 4.3.4.

Note that, as long as both the job stream and the server nodes are split equally among the sub-clusters, the traffic intensity at each sub-cluster is $\lambda/(N\nu)$, regardless of k_1. This is true also for trees with multiple levels of master nodes.

As λ increases, the range of possible values for k_1 given in (4.16) diminishes and eventually there will be no acceptable value of k_1. The 2-level network will be saturated but, as long as the server capacity is sufficient ($\lambda < N\nu$), a configuration with 3 levels of master nodes may be able to cope with the demand. Assign k_1 nodes to level 2; each of them controls k_2 masters at level 3, each of which is in turn responsible for $N/(k_1 k_2)$ servers.

The requirement that all master nodes are stable restricts the possible configurations. For each value of k_1 in the range $2 \leq k_1 < c_1/\lambda$, k_2 must

satisfy the inequalities

$$\frac{1}{k_1}\sqrt{\frac{\lambda N}{c_3}} < k_2 < \frac{c_2 k_1}{\lambda} . \tag{4.18}$$

Again, for each feasible pair (k_1, k_2), one can evaluate the corresponding average response time

$$W = W_1 + T_1 + W_2 + T_2 + W_3 + S_3 . \tag{4.19}$$

These values are compared to find the minimum response time, and hence the optimal network configuration for $m = 3$.

This process can be generalized for the computation of optimal values for $k_1, k_2, \ldots, k_{m-1}$ for a system with m levels of master nodes. The feasible values for k_1 are $2 \leq k_1 < c_1/\lambda$, and those for k_r $(r = 2, 3, \ldots, m-1)$ must satisfy the inequalities

$$\frac{1}{\prod_{j=1}^{r-1} k_j}\sqrt{\frac{\lambda N}{c_{r+1}}} < k_r < \frac{c_r \prod_{j=1}^{r-1} k_j}{\lambda} . \tag{4.20}$$

4.3.3 A simple heuristic

The number of feasible configurations of an m-level network is likely to be large. Once again, evaluating and comparing the average response times for each of these configurations will be time-consuming, and hence the simple heuristic for networks with $m = 2$ from section 4.2.3 may be extended.

The idea is to (a) restrict consideration to trees where all master node levels except the last are binary, and (b) choose the configuration of the last

master node level so as to minimize the total load on the master nodes visited by a job traversing the tree.

Step (a) consists in finding the smallest $m > 1$, such that $k_r = 2$ for $r < m - 1$ and there are integers that satisfy (4.20) for $r = m - 1$.

Consider, for example, the case where the range (4.16) is non-empty, i.e. $m = 2$ (the set of binary levels is empty). Now, for a given k_1, the total load, $\rho(k_1)$, of the level 1 master and a level 2 master is equal to

$$\rho(k_1) = \frac{\lambda k_1}{c_1} + \frac{\lambda N}{c_2 {k_1}^2} . \tag{4.21}$$

This total load may be minimized with respect to k_1 (the load at the service sub-cluster is not included because it does not depend on k_1). Differentiating (4.21) and equating to zero yields

$$k_1 = \sqrt[3]{\frac{2Nc_1}{c_2}} . \tag{4.22}$$

Choosing the nearest integer to the right-hand side of (4.22) as the value for k_1 matches the numerically obtained optimum quite closely, particularly when the network is heavily loaded.

If step (a) produces $m > 2$, then the master nodes in levels $1, \ldots, m-1$ form a binary tree, while each node in level $m-1$ has k_{m-1} dependents. The load of nodes at levels $1, \ldots, m-2$ does not depend on k_{m-1}. The total load of a level $m-1$ master and a level m master, $\rho(k_{m-1})$, is given by:

$$\rho(k_{m-1}) = \frac{\lambda k_{m-1}}{2^{m-2} c_{m-1}} + \frac{\lambda N}{(2^{m-2} k_{m-1})^2 c_m} . \tag{4.23}$$

Minimizing his total load with respect to k_{m-1} leads to

$$k_{m-1} = \sqrt[3]{\frac{Nc_{m-1}}{2^{m-3}c_m}} . \tag{4.24}$$

An attractive feature of this heuristic is that it does not depend on the arrival rate λ; the configuration is determined only by the master node processing parameters c_i. The trees constructed by the heuristic contain at least 2 levels of master nodes. The optimal configuration may not belong to that class, but the experiments in the next section show that the loss in performance incurred by using the heuristic is not large.

4.3.4 Results

Before proceeding with the numerical comparison of different configurations, the quality of the approximations employed is evaluated by simulation. The first experiment compares three routing policies among master nodes: uniform random, shortest queue and cyclic. The simulated configuration consists of 2 levels of master nodes ($m = 2$), with 3 nodes at level 2, each controlling a sub-cluster of 32 servers (a total of 100 nodes). The network parameters are $c_1 = 50, c_2 = 100, T_1 = 0.001$ and $\nu = 0.05$.

The simulation results are shown in figure 4.7. The performance measure, i.e. the average response time of a job, is plotted against the total offered load, $\lambda/(N\nu)$; the latter is varied by increasing λ. The figure confirms that both the shortest queue and cyclic policies perform significantly better than random routing; moreover, the difference between the former two is small under normal loading conditions (the two graphs begin to diverge only when

the offered load exceeds 0.9). Hence, the more easily implementable and analyzable cyclic routing policy may be used as a reasonable approximation to the shortest queue policy.

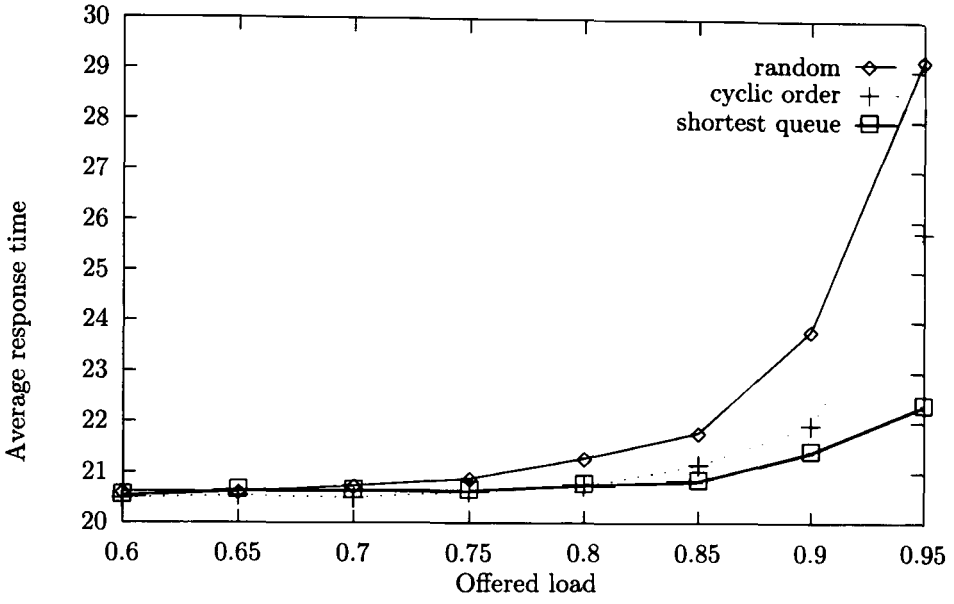

Figure 4.7: Comparison of different routing policies: $m = 2$ $N = 100$, $k_1 = 3$, $c_1 = 50$, $c_2 = 100$, $T_1 = 0.001$, $\nu = 0.05$

The next two experiments evaluate the approximation introduced by assuming that, under the cyclic routing policy, the *departures* from each master node are Poisson. Figure 4.8 compares the simulated total average response times with those computed using the GI/M/1 model (with Erlang interarrival intervals) for master nodes at levels higher than 1, and the M/M/n model at service sub-clusters. The network parameters are $m = 2$, $N = 100$, $c_i = 100$, $T_i = 0.001$ $(i = 1, 2)$ and $\nu = 0.1$. There are $k_1 = 5$ master nodes at level 2 and hence each service sub-cluster contains 20 servers. The average response times are plotted against the offered load, $\lambda/(N\nu)$, as λ increases.

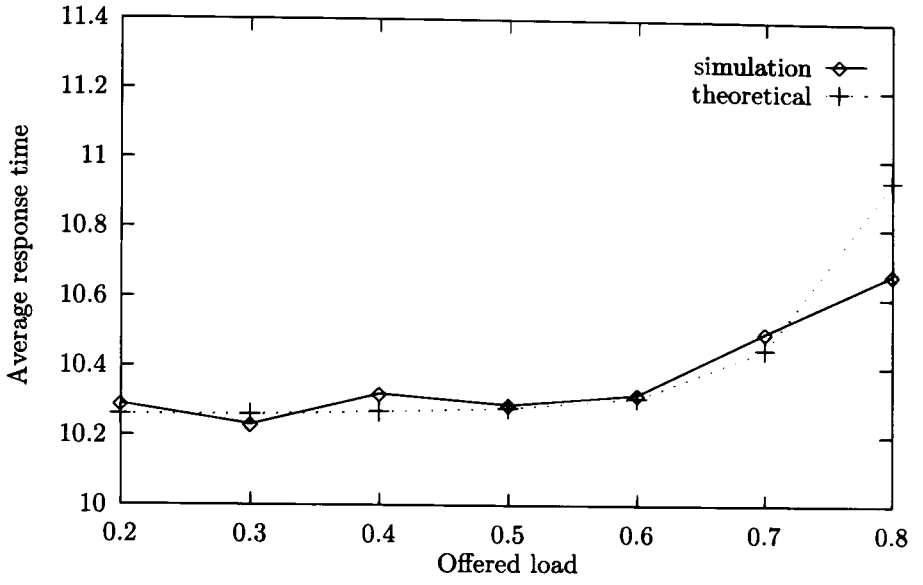

Figure 4.8: Comparison of theory and simulation: $m = 2$ $N = 100,\ k_1 = 5,\ c_i = 100,\ T_i = 0.001,\ \nu = 0.1$

A similar experiment is illustrated in Figure 4.9 for a network with three levels and twice as many service nodes: $m = 3$, $N = 200$, $c_i = 100$, $T_i = 0.001$, $\nu = 0.1$ There are 2 master nodes at level 2 and 5 master nodes at level 3. Again, each service sub-cluster contains 20 servers.

Both sets of results demonstrate good agreement between the simulated and approximated performance measures, except in very heavily loaded systems. It is to be expected that each additional level of master nodes would make the approximation less accurate in heavy traffic. These results indicate that the approximations tend to be pessimistic; they overestimate the average response times. This is not really surprising, since the cyclic routing policy tends to make both the arrival and the departure processes more regular (in the sense of having a lower variance) than the Poisson process with the same mean. Hence, the assumption that the departure process is Poisson

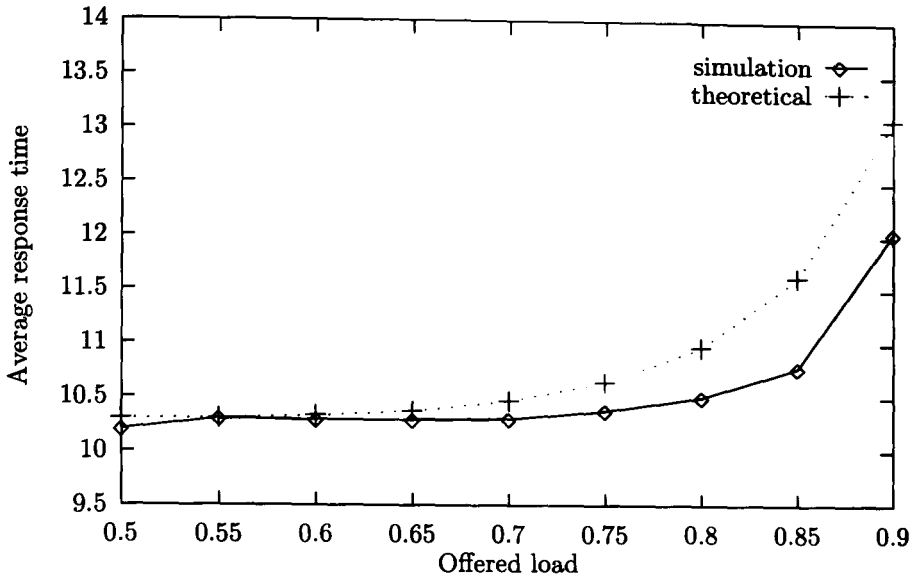

Figure 4.9: Comparison of theory and simulation: $m = 3$ $N = 200,\ k_1 = 2,\ k_2 = 5,\ c_i = 100,\ T_i = 0.001,\ \nu = 0.1$

tends to overestimate the variance.

From now on, the numerical approximation of the cyclic routing policy is used to evaluate and compare the performance of different network configurations, and to examine the quality of the simple configuration heuristic. The next set of experiments concern a system with 3 levels of master nodes ($m = 3$) as shown in figure 4.6. Both k_1 and k_2 may be varied in order to minimize the average response time. The number of service nodes is fixed, $N = 200$, as is their service rate, $\nu = 0.1$; the average transfer times between levels are equal, $T_1 = T_2 = 0.001$.

Figure 4.10 shows the average response times for different configurations, when the master node processing parameters are the same at all levels, $c_i = 100$ ($i = 1, 2, 3$). The arrival rate is $\lambda = 16$, i.e. the offered load is 0.8. A single level structure ($m = 1$) is not feasible, since $c_1 < \lambda N$. The only

feasible 2-level tree is the one with 6 master nodes at level 2. as indicated by (4.16). However, the performance of that tree is worse than several of the configurations for $m = 3$. The optimal 3-level configuration is $(k_1 = 2, k_2 = 4)$, although the configurations $(k_1 = 3, k_2 = 3)$ and $(k_1 = 2, k_2 = 5)$ are almost as good. The tree suggested by the heuristic (4.24) is the latter one: $(k_1 = 2, k_2 = 5)$.

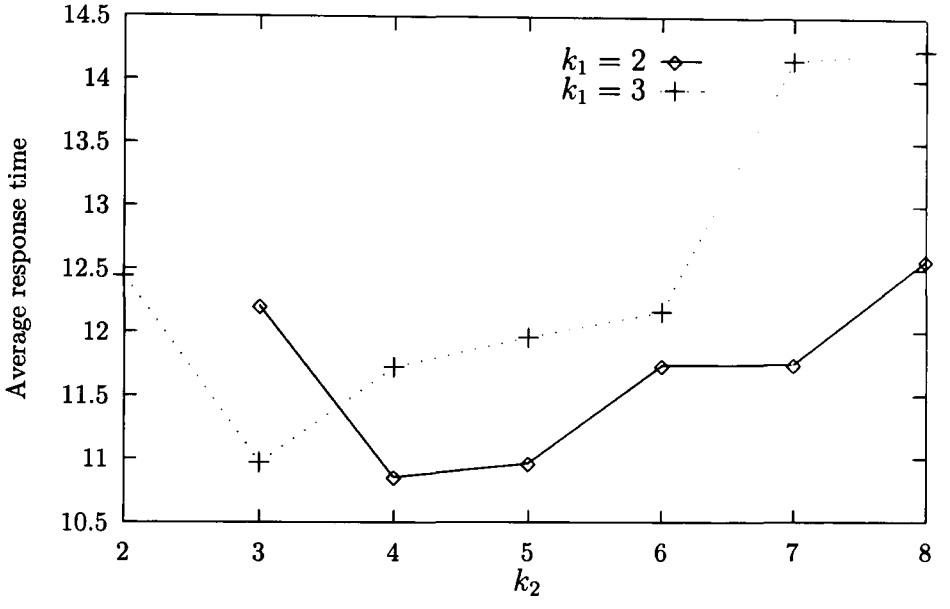

Figure 4.10: Performance of different configurations: $m = 3$
$N = 200$, $c_i = 100$, $T = 0.001$, $\lambda = 16$, $\nu = 0.1$

In figure 4.11, processing at levels 1 and 2 is faster, with $c_1 = c_2 = 200$, while that at level 3 is slower, $c_3 = 50$. The arrival rate is lower, $\lambda = 12$, giving an offered load of 60%. This system has a wider range of possible configurations. Again, an $m = 1$ structure is not feasible, but there are several 2-level configurations that are allowed by (4.16): $7 \leq k_1 \leq 16$. However, once again it is best is to add another level of master nodes and

consider 3-level trees. The optimal configuration is $(k_1 = 2, k_2 = 7)$. There are other trees, e.g. $(k_1 = 3, k_2 = 4, 5, 6)$ or $(k_1 = 2, k_2 = 6, 8, 9, 10)$, which are almost as good. The heuristic (4.24) produces one of the latter: $(k_1 = 2, k_2 = 9)$.

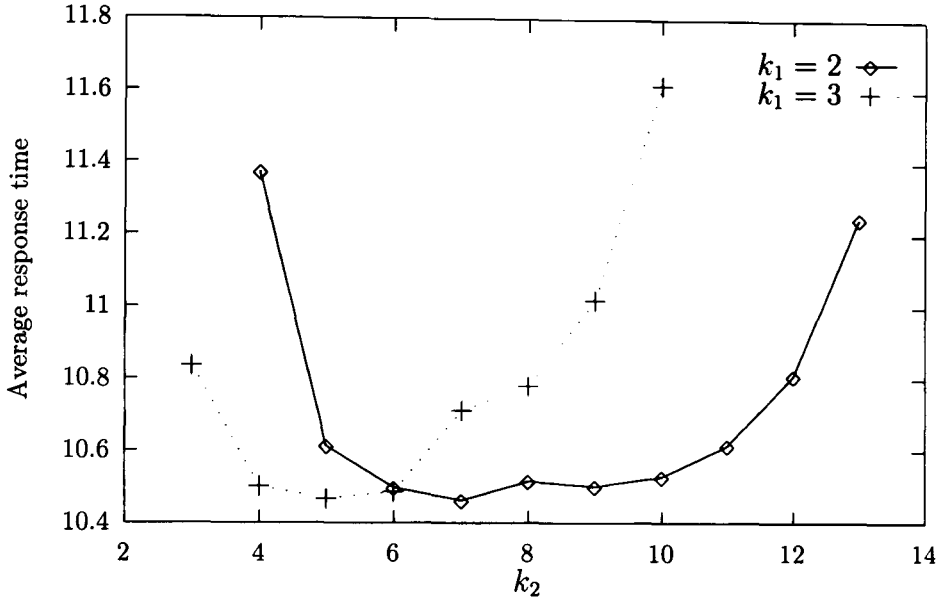

Figure 4.11: Performance of different configurations: $m = 3$ $N = 200$, $c_1 = c_2 = 200$, $c_3 = 50$, $T = 0.001$, $\lambda = 12$, $\nu = 0.1$

Thus, both examples illustrated in figures 4.10 and 4.11 show that, even though feasible network configurations exist with only 2 levels of master nodes, better performance may be obtained by 3-level trees.

In figure 4.12, processing at levels 1 and 2 is slower, with $c_1 = c_2 = 100$, while that at level 3 is significantly faster, $c_3 = 1000$. This could represent a wide network of routers forwarding jobs to a local cluster for fast processing. The arrival rate is again $\lambda = 12$, giving an offered load of 60%. This system also has a number of possible configurations, although the range is now not

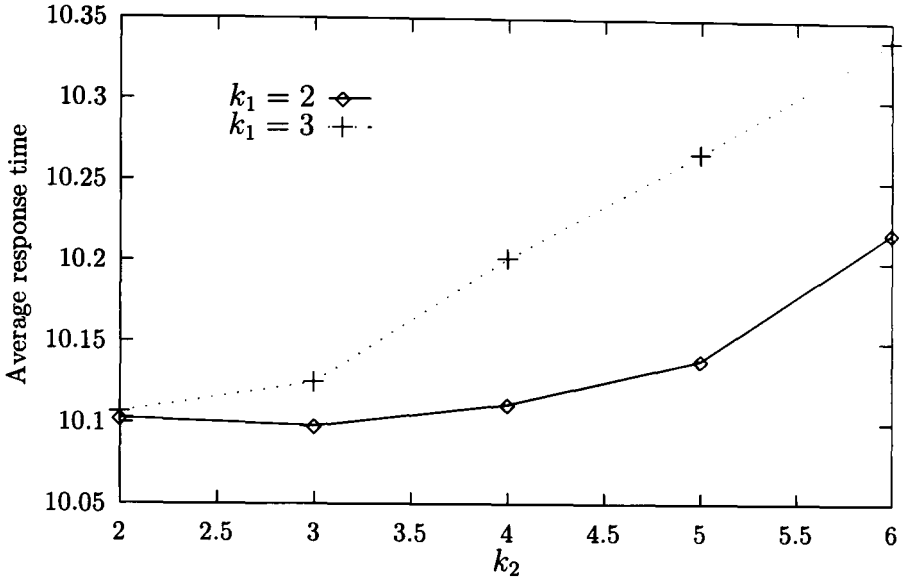

Figure 4.12: Performance of different configurations: $m = 3$ $N = 200,\ c_1 = c_2 = 100,\ c_3 = 1000,\ T = 0.001,\ \lambda = 12,\ \nu = 0.1$

so wide as for the system shown in figure 4.11. An $m = 1$ structure is still not feasible. Possible 2-level configurations allowed by (4.16) are: $2 \leq k_1 \leq 6$. However, once again it is best is to add another level of master nodes and consider 3-level trees, even when the routing decisions at levels 1 and 2 are comparatively slow. The optimal configuration is $(k_1 = 2, k_2 = 3)$. There are a few trees, e.g. $(k_1 = 2, k_2 = 2, 4)$ or $(k_1 = 3, k_2 = 2, 3)$, which are almost as good. The heuristic (4.24) produces one of the latter: $(k_1 = 2, k_2 = 2)$.

Since the heuristic allocation provided by (4.24) does not depend on the offered load, it is interesting to examine whether that allocation remains good at different loads. This is done in table 4.3.4, where the heuristic and optimal allocations are compared at offered loads varying from 0.2 to 0.9. The network has 200 service nodes and 3 levels of master nodes, with slower processing at levels 1 and 2 ($c_1 = c_2 = 50$), and faster processing at level 3

($c_3 = 100$).

In this example, the heuristic configuration not only performs well over a wide range of loads; it in fact becomes optimal when the load reaches or exceeds 60%.

	load	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
optimal	k_1	3	3	3	3	2	2	2	2
	k_2	5	5	5	5	4	4	4	4
heuristic	k_1	2	2	2	2	2	2	2	2
	k_2	4	4	4	4	4	4	4	4

Table 4.1: Optimal and heuristic configurations: $m = 3, N = 200$
$c_1 = c_2 = 50,\ c_3 = 100,\ T_i = 0.001,\ \nu = 0.1$

4.3.5 Conclusions

This chapter has demonstrated that the topological structure of a service network has a significant effect on its performance. For a given set of parameters, the optimal structure can be obtained numerically, using existing analytical results which apply to either random or cyclic routing policies. A simple heuristic which avoids the necessity of searching through all feasible configurations has been derived.

Chapter 5

Breakdowns and Repairs

5.1 Introduction

In service provisioning systems such as a Computing Grid, the quality of service, and the cost of providing it, is important both to the users and to the provider. A problem of particular interest is the effect that server breakdowns and other outages have on the performance of the system. That problem is the topic of this chapter.

Now under consideration is a model where demands, or *jobs*, arrive in a Poisson stream into a common queue and are served by a number, N, of servers in parallel. Each server goes through alternating periods of being *operative* and *inoperative*, independently of the others; the events causing a change of server state will be referred to as *breakdowns* and *repairs*, although in practice they may have other causes (e.g., scheduled maintenance, or tasks of higher priority). A system of this type is illustrated in figure 5.1.

Among the questions one may wish to ask in this context are:

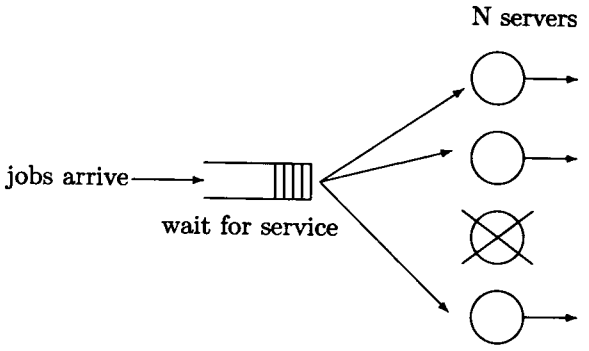

Figure 5.1: A multi-server system with breakdowns and repairs

1. How does the system perform? A common performance measure is the average response time or, equivalently, the average number of jobs present.

2. What is the minimum number of servers that would ensure a desired level of performance?

3. If there is a trade-off between the cost of making jobs wait and that of providing servers, what is the optimal number of servers that should be used?

The answers to all those questions depend not only on rates of demand and service, but also on the nature of the operative and inoperative intervals. Moreover, that 'nature' encompasses not only the means, but also the distributions of those random variables. Exactly how one should model breakdowns and repairs is a point that has not received much attention in the literature. There are several papers on the subject of multi-server queues with service interruptions (e.g., see [5, 9, 36, 38, 46]). However, they all make the assumption that both the operative and inoperative intervals are distributed

exponentially. The validity of that assumption has not been investigated.

The contribution of this research is two-fold: First, a large real-life data set is analyzed containing information about server breakdowns and repairs; this data has been collected and made available to us by Sun Microsystems. The results indicate that the distribution of repair times is reasonably close to an exponential, but that of the operative intervals is not. A good approximation for the latter is the hyperexponential distribution.

Second, how to obtain an exact solution for a model with non-exponentially distributed operative and/or inoperative intervals is demonstrated. This is done by reducing the problem to a Markov-modulated queue (with a suitably defined Markovian environment) and then solving it by spectral expansion. The solution can be computationally expensive and prone to numerical difficulties when the number of servers (and/or the number of phases in the hyperexponential distributions) is large. In those cases, one can apply a simple approximation whose accuracy improves when the offered load increases.

The statistical analysis of the Sun data set, including the fitting and testing of hyperexponential distributions, is described in section 5.2. The mathematical model based on these distributions, together with its exact and approximate solutions, is presented in section 5.3. Some numerical results illustrating the effects of different parameters on performance and costs are described in section 5.4. Section 5.5 contains a summary and brief conclusions.

5.2 The data set

The Sun Microsystems data set contains 140,000 rows of data, each row giving details of a particular *event*, corresponding to a server breakdown. Of immediate interest were the fields representing the time a server was inoperative, referred to as *Outage Duration*, and the time between a server breakdown and its next breakdown, referred to as *Time Between Events*. The lengths of operative periods can then be calculated as illustrated in figure 5.2.

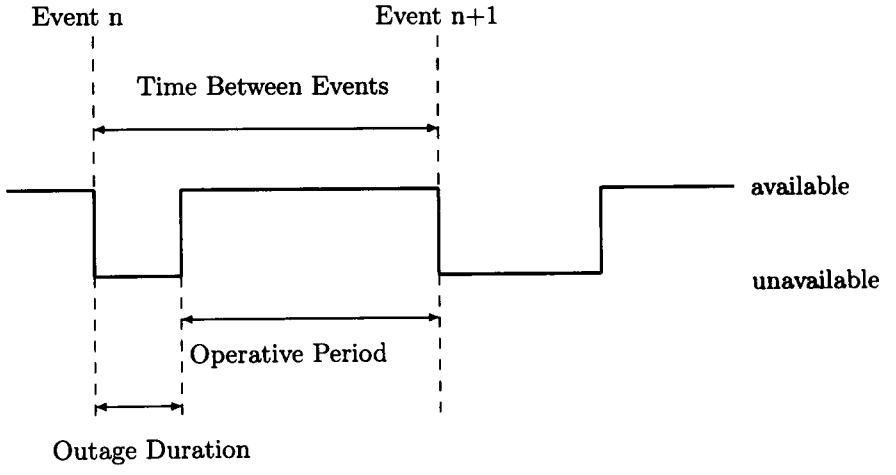

Figure 5.2: Alternating Periods of availability and unavailability

A small proportion of the data set (less than 4%) contained anomalous entries (Time Between Events was smaller than the Outage Duration). This data was ignored. Empirical probability density functions (histograms) were generated for both the operative and inoperative periods, by grouping observed period lengths into appropriate intervals.

Consider the operative periods. If the i^{th} observation interval has a mid-

point x_i, and f_i of the observed operative periods fall into that interval, then the corresponding empirical density, d_i, is obtained by assuming that the operative periods take value x_i with probability $p_i = f_i/n$, where n is the total number of observations; then $d_i = p_i/\delta_i$, where δ_i is the length of the i^{th} interval. A similar procedure is followed for the inoperative periods.

The empirical densities of the operative and inoperative periods are shown in figures 5.3 and 5.5, respectively (together with the fitted hypothetical distributions, to be described below). In each case, the observed range of values was divided into intervals of equal length. The time unit has been deliberately omitted, for reasons of confidentiality.

The two empirical densities were used to derive estimates for the moments of the corresponding distributions. The k^{th} estimated moment, $\tilde{M}_k$, is calculated as

$$\tilde{M}_k = \sum x_i^k p_i \ ; \ k = 1, 2, \ldots \ , \tag{5.1}$$

where the sum extends over all empirical values.

The estimated variance, $\tilde{V}$, and coefficient of variation, $\tilde{C}^2$, are given by

$$\tilde{V} = \tilde{M}_2 - \tilde{M}_1^2 \ ; \ \tilde{C}^2 = \frac{\tilde{M}_2}{\tilde{M}_1^2} - 1 \ . \tag{5.2}$$

From the empirical densities one can also obtain the empirical cumulative distribution functions,

$$\tilde{F}(x_i) = \sum_{j=1}^{i} p_j \ . \tag{5.3}$$

The null hypothesis that an empirical cumulative distribution function, $\tilde{F}(x)$, is consistent with a given hypothetical one, $F(x)$, can be tested by means

of the Kolmogorov-Smirnov *goodness-of-fit* test, [21]. The hypothesis is accepted if the value of the statistic D, calculated as

$$D = \max_{x_i} \left| F(x_i) - \tilde{F}(x_i) \right| , \tag{5.4}$$

is sufficiently small, for a given level of significance; otherwise it is rejected (the higher the level of significance, the more difficult it is to pass the test).

On the basis of the Sun data set, the hypothesis that the server operative periods are distributed exponentially with a mean obtained from the sample, is strongly rejected. The calculated value of the Kolmogorov-Smirnov statistic, using 50 points x_i, was $D = 0.4742$; it would have had to have been less than 0.19 to pass the test at 5% significance, and less than 0.23 at 1% significance.

The inoperative intervals are more likely to be exponentially distributed: that hypothesis also fails the Kolmogorov-Smirnov test, but not so badly. Moreover, it shall be seen later that an exponential distribution with a slightly different mean passes the test quite comfortably.

The next task is to find hypothetical distributions that do agree with the empirical densities for the operative and inoperative periods. An indication of where to look is provided by the values of the estimated coefficients of variation, which are both greater than 1 ($\tilde{C}^2 = 4.6$ for the operative periods). This suggests that the family of hyperexponential distributions, all of which have coefficients of variation greater than 1, may be a good place to start. An n-phase hyperexponential density function is a linear combination of n

exponential densities with different parameters; it has the form

$$f(x) = \sum_{j=1}^{n} \alpha_j \xi_j e^{-\xi_j x} \;\; ; \;\; \alpha_j, \xi_j > 0 \;\; ; \;\; \sum_{j=1}^{n} \alpha_j = 1 \; . \tag{5.5}$$

Such a density is defined by $2n-1$ parameters: n 'rates' ξ_j and $n-1$ 'weights' α_j, (the last weight is given by the normalizing condition in (5.5)). Hence, an n-phase hyperexponential distribution is completely determined by its first $2n-1$ moments. Those moments are expressed in terms of the parameters as follows:

$$M_k = \sum_{j=1}^{n} \frac{k! \alpha_j}{\xi_j^k} \;\; ; \;\; k = 1, 2, \ldots 2n-1 \; . \tag{5.6}$$

Thus, a hyperexponential distribution can be fitted to a given empirical density by first choosing the number of phases, n, and then determining the parameters α_j and ξ_j so that

$$M_k = \tilde{M}_k \;\; ; \;\; k = 1, 2, \ldots 2n-1 \; . \tag{5.7}$$

The above procedure was carried out for the operative periods, with $n = 3$. The empirical density provided the first 5 estimated moments, $\tilde{M}_1, \ldots, \tilde{M}_5$. However, it turned out that the task of solving (5.7) is computationally difficult, because those equations are highly non-linear. Iterative methods such as Newton or Gauss-Seidel [39] failed to converge. Instead, the weights α_j were eliminated explicitly from the first two equations in (5.7), and a

brute force search was used to find the rates ξ_j that minimize

$$\min_{\xi_1,\xi_2,\xi_3} \sum_{k=3}^{5} |M_k - \tilde{M}_k| \,. \tag{5.8}$$

It was observed that two of the rates thus calculated were almost equal. In other words, a 2-phase hyperexponential distribution fits the data as well as a 3-phase one (re-running the Gauss-Seidel iterations for $n = 2$ resulted in convergence).

The 2-phase hyperexponential distribution that provides the best fit to the empirical density has parameters $\alpha_1 = 0.7246$, $\alpha_2 = 0.2754$, $\xi_1 = 0.1663$ and $\xi_2 = 0.0091$. That is, approximately 72% of the operative periods are distributed exponentially with mean 6, and 28% of them are distributed exponentially with mean 110. That density is shown together with the empirical one in figure 5.3. Also illustrated is the density of an exponential distribution with the same mean (i.e. mean $1/\xi = \alpha_1/\xi_1 + \alpha_2/\xi_2 = 34.62$) . The exponential distribution is observed to be a poor fit, and indeed fails the Kolmogorov-Smirnov goodness-of-fit test badly at the 5% significance level. The density of the hyperexponential distribution passes the Kolmogorov-Smirnov goodness-of-fit test at level of significance 5%, and also at 10% (the calculated statistic with 50 points x_i has value $D = 0.1412$, whereas the 5% and 10% critical values are 0.19 and 0.17 respectively). The cumulative distributions used for this latter Kolmogorov-Smirnov test may be seen in figure 5.4.

Similarly, for the inoperative periods, a best-fit 2-phase hyperexponential distribution was found with weights $\beta_1 = 0.9303$ and $\beta_2 = 0.0697$, and rates

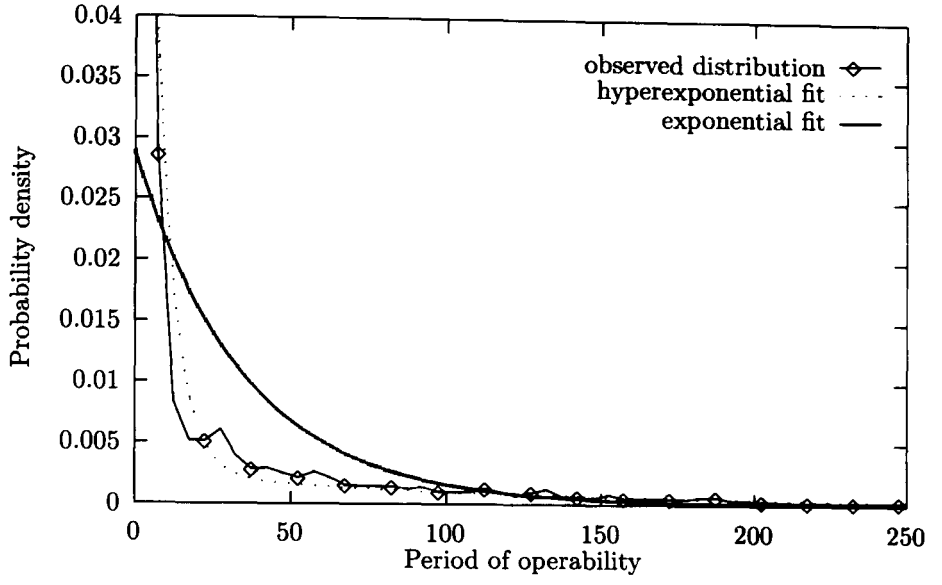

Figure 5.3: Densities of operative periods (0 – 250)

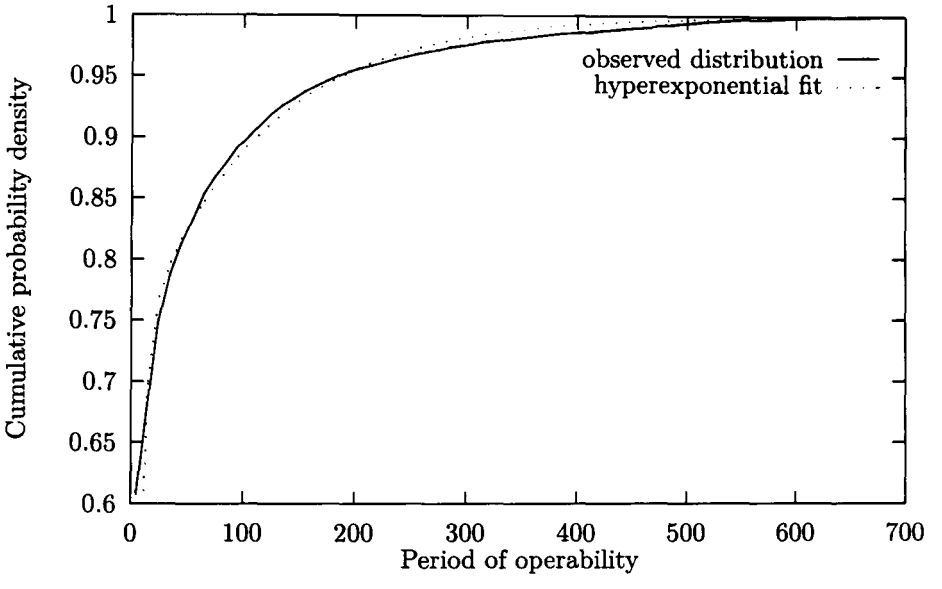

Figure 5.4: Cumulative distribution of operative periods

$\eta_1 = 25.0043$ and $\eta_2 = 1.6346$. This represents a mixture where approximately 93% of the inoperative periods are distributed exponentially with

mean 0.04 and 7% are distributed exponentially with mean 0.61. The fitted density, together with the empirical one, is shown in figure 5.5. It passes the Kolmogorov-Smirnov test at both the 5% and the 10% level of significance (the calculated statistic with 40 points x_i is $D = 0.1832$; the 5% and 10% critical values are 0.21 and 0.19 respectively). The cumulative distributions used for this Kolmogorov-Smirnov test may be seen in figure 5.6.

In view of the fact that the second component of the fitted hyperexponential distribution contributes very little to the mixture, it is not unreasonable to model the inoperative periods as being distributed exponentially. Indeed, the first component on its own, i.e. the exponential distribution with mean 0.04, passes the Kolmogorov-Smirnov test at level 5% (fails, but not badly, at 10%).

A more detailed examination of the Sun data set revealed certain non-random phenomena, e.g. spikes due to regular weekend maintenance outages. A second analysis of the data was carried out, with these regularities removed. It turned out that the same conclusions were reached regarding the nature of the underlying distributions.

5.3 The model and its solution

The preceding section offers evidence that, in a realistic model of a multi-server system with breakdowns and repairs, it is appropriate to assume that the distribution of the operative periods is hyperexponential, with n phases and suitably chosen weight and rate parameters α_j and ξ_j $(j = 1, 2, \ldots, n)$. Similarly, the inoperative periods can be assumed to have a hyperexponential

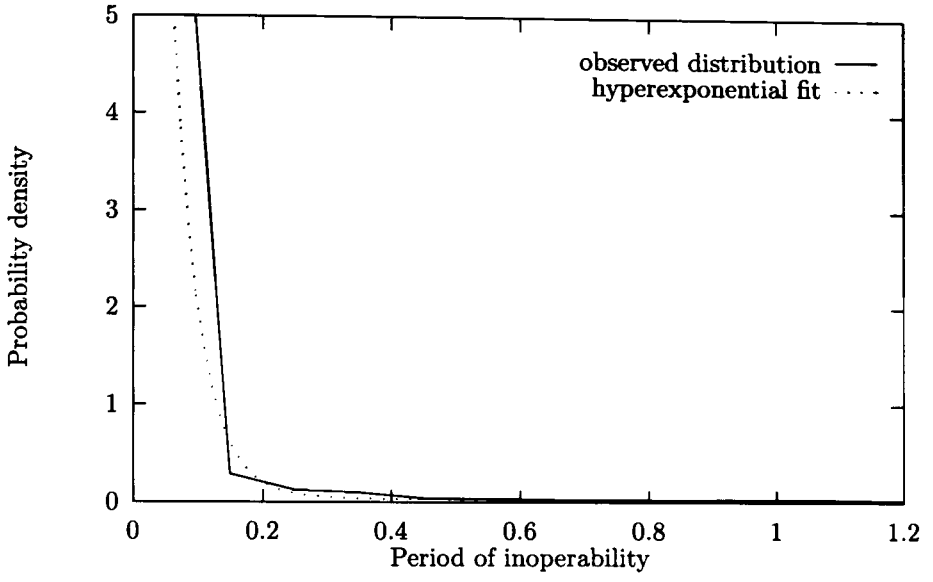

Figure 5.5: Densities of Inoperative periods $(0 - 1.2)$

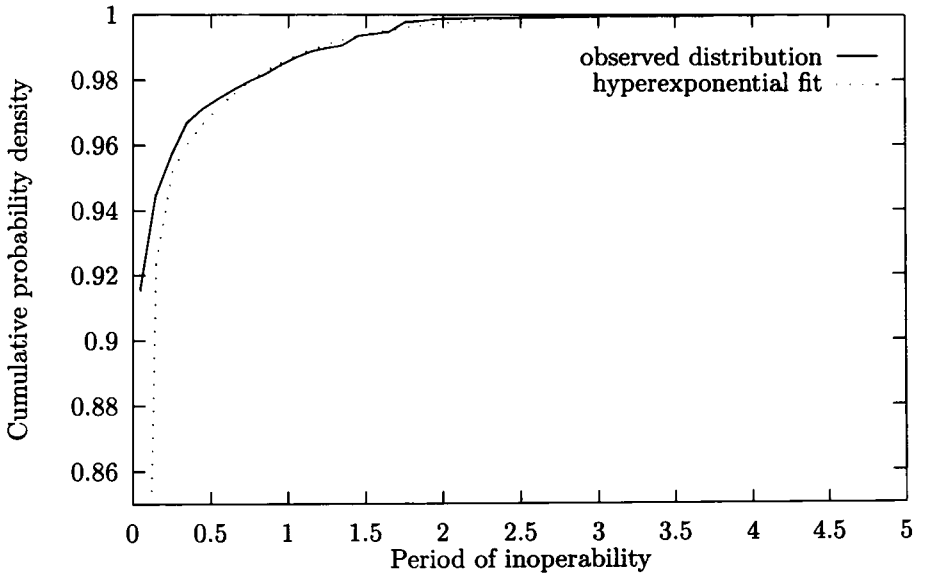

Figure 5.6: Cumulative distribution of inoperative periods

distribution with m phases and weight and rate parameters β_k and η_k ($k = 1, 2, \ldots, m$).

Assume further that jobs arrive according to a Poisson process with rate λ, and their required service times are distributed exponentially with mean $1/\mu$. The queue is unbounded. All arrival, service, breakdown and repair events are mutually independent. An operative server cannot be idle if there are jobs waiting to be served. A job whose service is interrupted by a server breakdown is returned to the front of the queue. When an operative server becomes available, the service is resumed from the point of interruption, without any switching overheads.

The above assumptions ensure that the system is modelled by a Markov process whose state at any moment in time is described by a triple $S = (\mathbf{X}, \mathbf{Y}, Z)$. Here, $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ is a vector whose j^{th} element, x_j, indicates how many servers are in phase j of an operative period; the number of operative servers is $x = x_1 + x_2 + \ldots + x_n$. Similarly, $\mathbf{Y}$ is a vector whose k^{th} element, y_k, indicates how many servers are in phase k of an inoperative period; the number of inoperative servers is $y = y_1 + y_2 + \ldots + y_m$. Finally, Z is the number of jobs present. The valid states must of course satisfy $x + y = N$, where N is the total number of servers.

The instantaneous transition rates from state $S = (\mathbf{X}, \mathbf{Y}, Z)$ to state

$S' = (\mathbf{X}', \mathbf{Y}', Z')$ are equal to

$$r(S, S') = \begin{cases} \lambda & \text{if } Z' = Z + 1 \\ \min(Z, x)\mu & \text{if } Z' = Z - 1 \\ x_j \xi_j \beta_k & \text{if } x'_j = x_j - 1,\ y'_k = y_k + 1 \\ y_k \eta_k \alpha_j & \text{if } x'_j = x_j + 1,\ y'_k = y_k - 1 \\ 0 & \text{otherwise} \end{cases} , \tag{5.9}$$

where all state variables that have not been mentioned have the same values in S and S'.

The stability condition for this queue has a simple form. Denote the average lengths of the operative and inoperative periods by $1/\xi$ and $1/\eta$, respectively. They are given by

$$\frac{1}{\xi} = \sum_{j=1}^{n} \frac{\alpha_j}{\xi_j} \ ; \ \frac{1}{\eta} = \sum_{k=1}^{m} \frac{\beta_k}{\eta_k} . \tag{5.10}$$

The long-term fraction of time that a given server is operative is equal to $\eta/(\xi + \eta)$. Hence, the steady state average number of operative servers is $N\eta/(\xi + \eta)$; this is independent of the queue of jobs. That queue is stable iff the offered load is less than the average number of operative servers, i.e.

$$\frac{\lambda}{\mu} < \frac{N\eta}{\xi + \eta} . \tag{5.11}$$

Note that this condition depends only on the averages of the operative and inoperative periods; not on their distribution. However, the queue size distribution, and hence the measures of performance, depend very much on the

distributions of operative and inoperative periods. Computing those performance measures is the next task.

5.3.1 Spectral expansion solution

The model defined here is a special case of a *Markov-modulated queue*, i.e., a queue whose arrival and/or service parameters depend on the state of a Markovian environment. In this case, the state of the environment is described by the vectors $\mathbf{X}$ and $\mathbf{Y}$, specifying the numbers of servers in each of the possible operative/inoperative states. The environment affects the queue via the number of operative servers, which determines the departure rate (second line in the right-hand side of (5.9)).

Markov-modulated queues can be solved by the method of 'Spectral Expansion' (e.g., see [37]).

The number, s, of different environment states, is equal to the number of ways that the integer N can be partitioned into a sum of $n+m$ components. This is equivalent to the number of ways that N indistinguishable balls may be allocated into $n+m$ distinguishable boxes [45]. That number is

$$s = \left(\begin{array}{c} N+n+m-1 \\ n+m-1 \end{array} \right) . \tag{5.12}$$

One can therefore enumerate the states of the environment using a single integer, i, called 'operational mode': $i = 0, 1, \ldots, s-1$. For example, in a system where $N = 2$, $n = 2$ and $m = 1$, the operational mode i may take 6

different values:

i=0: 2 inoperative servers

i=1: 1 operative in phase 1 and 1 inoperative

i=2: 1 operative in phase 2 and 1 inoperative

i=3: 2 operative in phase 1

i=4: 1 operative in phase 1 and 1 operative in phase 2

i=5: 2 operative in phase 2

The system is then said to be in state (i, j) if the operational mode is i and there are j jobs present ($i = 0, 1, \ldots, s-1$; $j = 0, 1, \ldots$). The transition rates (5.9) can be expressed in terms of the following $s \times s$ matrices:

(a) Matrix A contains transition rates that change the operative mode but not the number of jobs: from state (i, j) to state (k, j) ($0 \leq i, k < s$, $i \neq k$). The main diagonal of A is zero by definition. In the above example, with $s = 6$, the matrix A has the form

$$A = \begin{bmatrix} 0 & 2\eta\alpha_1 & 2\eta\alpha_2 & 0 & 0 & 0 \\ \xi_1 & 0 & 0 & \eta\alpha_1 & \eta\alpha_2 & 0 \\ \xi_2 & 0 & 0 & 0 & \eta\alpha_1 & \eta\alpha_2 \\ 0 & 2\xi_1 & 0 & 0 & 0 & 0 \\ 0 & \xi_2 & \xi_1 & 0 & 0 & 0 \\ 0 & 0 & 2\xi_2 & 0 & 0 & 0 \end{bmatrix}$$

(b) Matrix B contains transitions that increase the number of jobs in the system by 1: from state (i, j) to state $(k, j+1)$. Since in this model arrivals

do not change the operational mode, B is diagonal:

$$B = \lambda I \ ,$$

where I is the identity matrix of order s.

(c) Matrices C_j contain transitions that decrease the number of jobs in the system by 1: from state (i,j) to state $(k,j-1)$. Since departures do not change the operational mode, C_j is diagonal. For the same example, C_j has the form

$$C_j = \begin{bmatrix} \mu_{0,j} & & & & & \\ & \mu_{1,j} & & & & \\ & & \mu_{1,j} & & & \\ & & & \mu_{2,j} & & \\ & & & & \mu_{2,j} & \\ & & & & & \mu_{2,j} \end{bmatrix}$$

where $\mu_{i,j} = min(i,j)\mu$; Note that these matrices depend on j if $j < N$, but cease to do so when $j \geq N$; then the index j may be dropped and $C_j = C$. Also, $C_0 = 0$ by definition.

Let $p_{i,j}$ be the steady state probability that the system is in state (i,j). Define also the row vectors of probabilities corresponding to states with j jobs in the system:

$$\mathbf{v}_j = (p_{0,j}, p_{1,j}, \ldots, p_{s-1,j}) \ ; \ j = 0, 1, \ldots \ . \tag{5.13}$$

Then, the balance equations for the equilibrium probabilities can be written

as:

$$\mathbf{v}_j[D^A + B + C_j] = \mathbf{v}_{j-1}B + \mathbf{v}_j A + \mathbf{v}_{j+1}C_{j+1} \ , \ j = 0, 1, \dots \ , \qquad (5.14)$$

where D^A is the diagonal matrix whose ith diagonal element is equal to the ith row sum of A.

When $j > N$, the matrices in equations (5.14) do not depend on j. The equations can be re-written in the form of a homogeneous vector difference equation of order 2:

$$\mathbf{v}_j Q_0 + \mathbf{v}_{j+1}Q_1 + \mathbf{v}_{j+2}Q_2 = \mathbf{0} \ ; \ j = N, N+1, \dots \ , \qquad (5.15)$$

where $Q_0 = B$, $Q_1 = A - D^A - B - C$ and $Q_2 = C$. Associated with equation (5.15) is the so-called 'characteristic matrix polynomial', $Q(z)$, defined as

$$Q(z) = Q_0 + Q_1 z + Q_2 z^2 \ . \qquad (5.16)$$

Let z_k be the 'generalized eigenvalues', of $Q(z)$ in the interior of the unit disk, and d be their number. Denote by $\mathbf{u}_k$ the corresponding 'generalized left eigenvectors'. These eigenvalues and eigenvectors satisfy

$$det[Q(z_k)] = 0 \ ; \ |z_k| < 1 \ ; \ k = 1, 2, \dots, d \ , \qquad (5.17)$$

where $det[Q(z)]$ is the determinant of $Q(z)$. Also,

$$\mathbf{u}_k Q(z_k) = \mathbf{0} \ ; \ k = 1, 2, \dots, d \ . \qquad (5.18)$$

In what follows, the qualification *generalized* will be omitted.

The theory of spectral expansion shows that, when the queue is ergodic, the number of eigenvalues in the interior of the unit disk is equal to the number of states of the Markovian environment. In the case here described, $d = s$. Moreover, experience indicates that they are simple. Then, the solution of (5.15) has the form

$$\mathbf{v}_j = \sum_{k=1}^{s} \gamma_k \mathbf{u}_k z_k^j \; ; \; j = N, N+1, \ldots \; , \tag{5.19}$$

where $\gamma_1, \ldots, \gamma_s$ are some (possibly complex) constants. Those coefficients, and the 'boundary' probabilities, $p_{i,j}$, for $j < N$, are determined from the balance equations (5.14), for $j = 0, 1, \ldots, N$. This is a set of $(N+1)s$ linear equations with Ns unknown probabilities (the vectors $\mathbf{v}_j$ for $j = 0, 1, \ldots, N-1$), plus the s constants γ_k. However, only $(N+1)s - 1$ of these equations are linearly independent, since the generator matrix of the Markov process is singular. On the other hand, an additional independent equation is provided by the requirement that all probabilities must sum up to 1:

$$\sum_{j=0}^{\infty} (\mathbf{v}_j \cdot \mathbf{1}) = 1 \; , \tag{5.20}$$

where $\mathbf{1}$ is a column vector with s elements, all of which are equal to 1.

5.3.2 A simple approximation

The exact solution is computationally intensive, and for systems with many operational modes (large number of servers and/or large number of phases

in the hyperexponential distributions), that solution may be intractable or prone to numerical problems. In that case, one may accept an approximate solution which is numerically robust and very simple to implement [34].

The approximation consists of discarding all terms in the spectral expansion solution (5.19), except the one corresponding to the eigenvalue with the largest modulus, z_s (which is always real and positive). That amounts to assuming that the queue size is distributed geometrically with parameter z_s, and is independent of the operational mode. The approximate solution has the form

$$\mathbf{v}_j = \frac{\mathbf{u}_s}{(\mathbf{u}_s \cdot \mathbf{1})}(1 - z_s)z_s^j \ ; \ j = 0, 1, \ldots \ . \tag{5.21}$$

It requires the computation of only one eigenvalue and its corresponding left eigenvector.

It has been shown (see [34]) that the geometric approximation is asymptotically exact when the system is heavily loaded.

5.4 Numerical results

The solution described in the previous section yields performance metrics which may be used to answer the questions raised in the Introduction. In particular, the average number of jobs present in the system, L, may be computed and hence the average response time, $W = L/\lambda$ (by Little's theorem). Moreover, if it costs c_1 per unit time to hold a job in the system, and c_2 per unit time to provide a server, then the steady state total cost, C, associated

with hosting a service cluster may be expressed as

$$C = c_1 L + c_2 N \, . \tag{5.22}$$

Such a cost function implies that there is a trade-off between the 'user' costs (measured by $c_1 L$), which decrease with N, and the 'provider' costs (measured by $c_2 N$), which increase with N. It can be expected that, for each set of parameters, there will be an optimal number of servers.

Several numerical experiments were carried out in the context of a system where the operative periods have a 2-phase hyperexponential distribution, while the inoperative periods are distributed exponentially (i.e., $n = 2$, $m = 1$). That is, the queue is modulated by an environment which, when there are N servers, has $s = (N+2)(N+1)/2$ operational modes. In all cases, the average required service time is $1/\mu = 1$.

In the first experiment, the parameters of the operative and inoperative periods are fixed as for the fitted distributions ($\alpha_1 = 0.7246, \xi_1 = 0.1663, \alpha_2 = 0.2754, \xi_2 = 0.0091, \eta = 25$), and N is varied.

Figure 5.7 shows how the cost function (5.22) changes with N, for three different values of the arrival rate. The values of the cost coefficients, $c_1 = 4, c_2 = 1$, reflect a situation where waiting is quite strongly discouraged. As expected, for each λ there is an optimal value of N that minimizes C. Moreover, the heavier the load, the larger the optimal N (the latter is 11 for $\lambda = 7$, 12 for $\lambda = 8$ and 13 for $\lambda = 8.5$).

The next experiment aims to evaluate the effect of operative period variability on performance. The average length of the operative period,

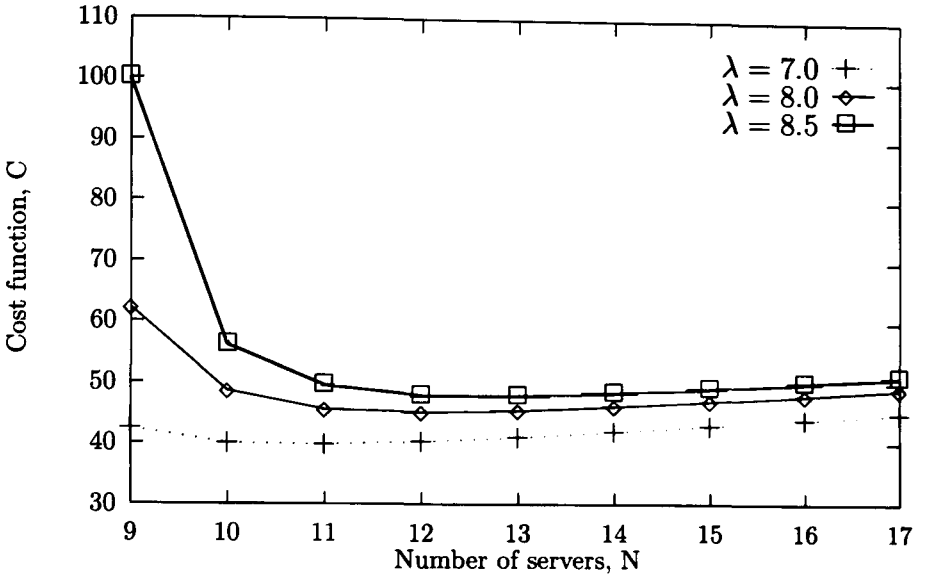

Figure 5.7: Cost as a function of N
$\alpha_1 = 0.7246, \xi_1 = 0.1663, \xi_2 = 0.0091, \eta = 25, c_1 = 4, c_2 = 1$

$1/\xi = \alpha_1/\xi_1 + \alpha_2/\xi_2$, is kept fixed at 34.62, but the coefficient of variation is varied by changing α_1, α_2 and ξ_1 (the operative periods in phase 1 become larger and less likely). In figure 5.8, L is plotted against C^2 for two different arrival rates, λ. The average inoperative period is fixed at $1/\eta = 5$. The value $C^2 = 1$ corresponds to the exponential distribution. The first point on each curve, where $C^2 = 0$ (i.e., constant operative periods) was obtained by simulation.

In all cases, the average queue size grows with the coefficient of variation. The effect is weak when the system is lightly loaded, but becomes more pronounced as the load increases. At heavy loads, an assumption of exponentially distributed operative periods can seriously underestimate the average queue size and hence the number of servers that are required in order to ensure a target quality of service.

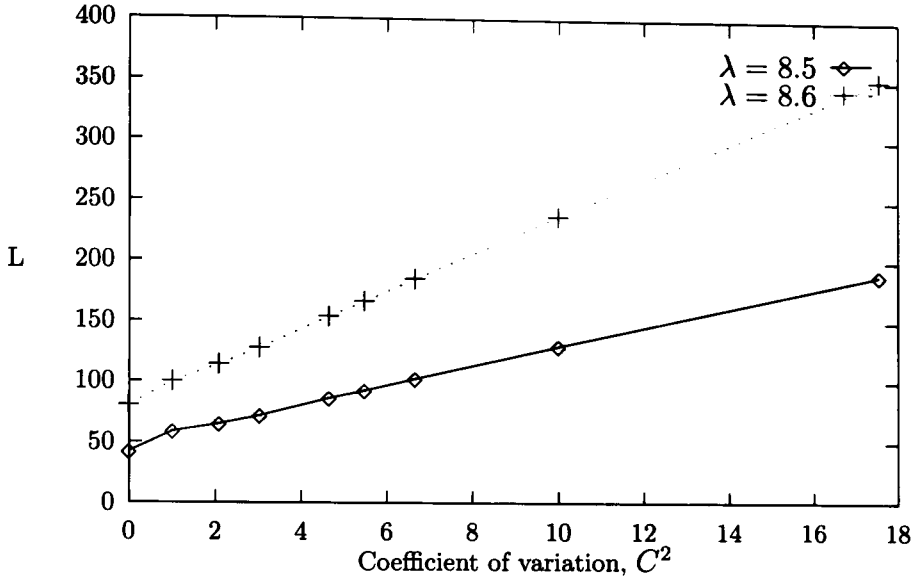

Figure 5.8: Average queue size against coefficient of variation
$N = 10, \eta = 0.2, \xi = 0.0289$

A similar effect is displayed in figure 5.9, where the distribution of the operative periods is kept fixed, while the availability of the servers is reduced by increasing the average inoperative period. The figure shows the average queue sizes under exponentially and hyperexponentially distributed operative periods with the same mean. The predictions corresponding to the exponential distribution are seen to become more and more over-optimistic as the average repair time increases.

The accuracy of the geometric approximation (5.21) is illustrated in figure 5.10. The average queue size is plotted against the arrival rate, for a system with 10 servers; the other parameters are the same as in figure 5.7. The figure confirms the theory that as the load increases, the approximation becomes more accurate.

The last experiment demonstrates how the model and its solution can be

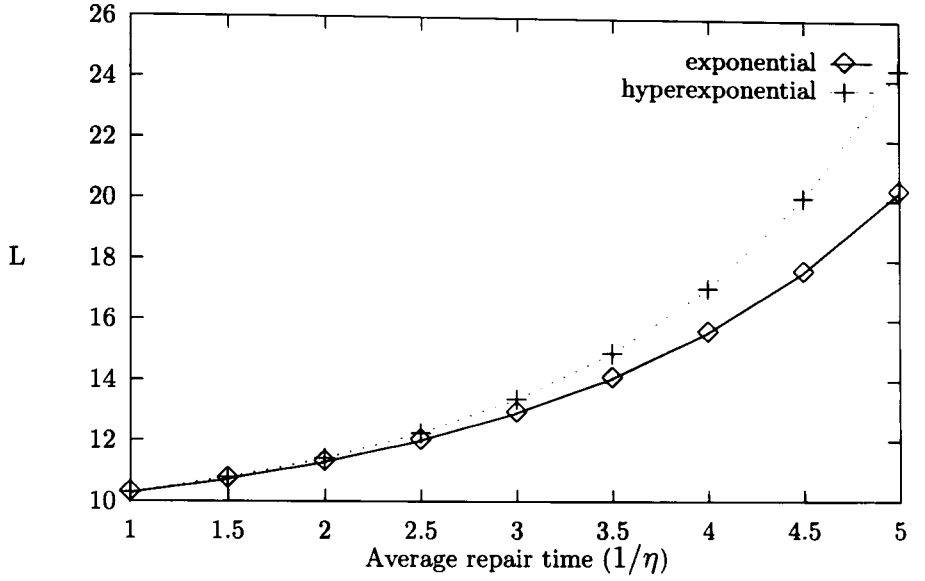

Figure 5.9: Average queue size against average repair time
$N = 10, \lambda = 8, \xi = 0.0289$

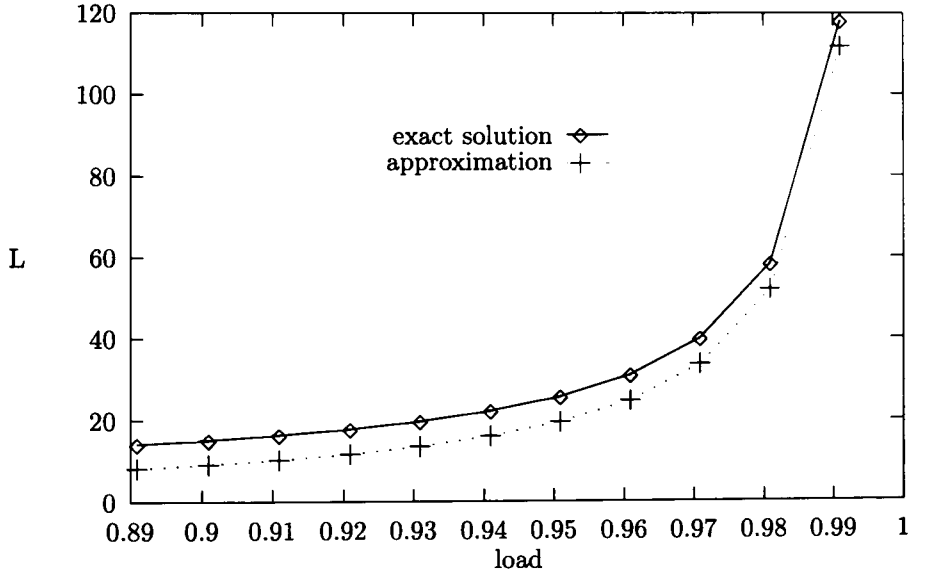

Figure 5.10: Exact and approximate solutions: increasing load
$N = 10, \alpha_1 = 0.7246, \xi_1 = 0.1663, \xi_2 = 0.0091, \eta = 25$

used to answer questions of the type "what is the minimum number of servers that would ensure a certain level of performance?". In figure 5.11, the average response time is plotted against the number of servers. The characteristics of the operative and inoperative periods are the same as in figure 5.7. Both the exact and the approximate solutions were evaluated. As an example of an application of such a figure, suppose that the objective was to ensure that the average response time does not exceed 1.5. The results would then indicate that at least 9 servers should be deployed. On this occasion the approximate solution underestimates the average response times; in other cases it overestimates them.

When N becomes large (greater than about 24), the exact solution begins to warn of possible numerical problems due to ill-conditioned matrices. The approximation does not display such problems.

5.5 Conclusions

In this chapter an attempt has been made to improve the realism of models used to evaluate and optimize multi-server systems subject to breakdowns and repairs. Statistical analysis of a large volume of data concerning real servers has shown that their operative periods are not distributed exponentially, but that a good fit can be obtained with a hyperexponential distribution. The inoperative periods may reasonably be assumed to have an exponential distribution, although a hyperexponential distribution would be more accurate for them too.

A model with hyperexponentially distributed operative and inoperative

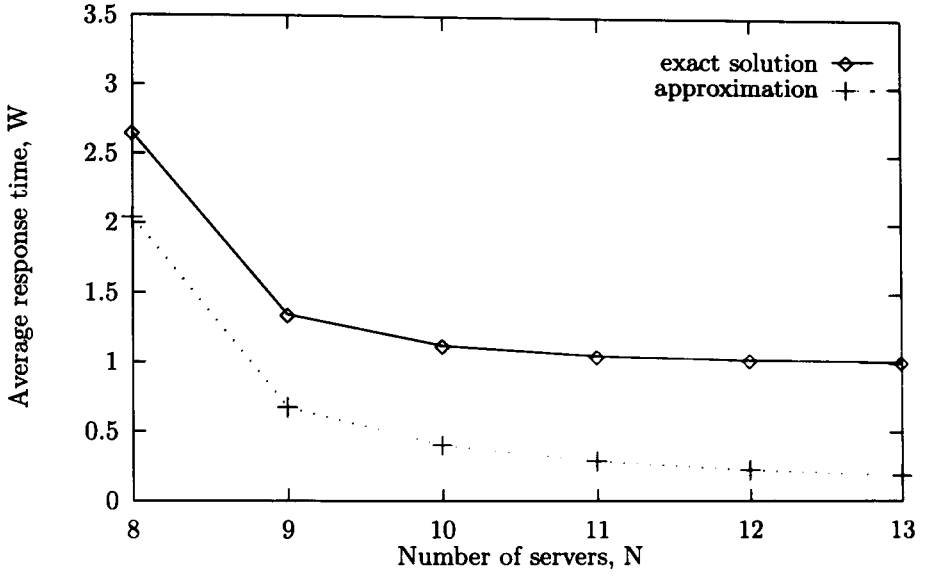

Figure 5.11: Average response time as a function of N
$\alpha_1 = 0.7246, \xi_1 = 0.1663, \xi_2 = 0.0091, \eta = 25, \lambda = 7.5$

periods has been formulated and solved exactly and approximately. These solutions have been used in numerical experiments, addressing a number of cost and quality of service problems.

Chapter 6

Conclusions and Future Work

It was proposed to determine strategies for maximising efficiency in service provisioning systems. To approach this, a number of models have been formulated and analysed in order to determine and evaluate a variety of operating strategies for the efficient deployment of computing resources. The analysis outlined in this thesis has been broken down into three distinct areas.

In chapters 2 and 3, the problem under examination was the dynamic reconfiguration of heterogeneous clusters of resources. The provision of a service involves the hosting of such clusters, and reconfiguring the resources to respond to changes in patterns of demand is of significant importance if such a provision is to be financially viable. The initial problem of interest outlined in chapter 2 was to limit the model to only two job types, with two associated clusters devoted to serving each of these job requests. The optimal reconfiguration policy for 2 job types was computed and tabulated. However, this computation was subject to complexity constraints imposed by the size of the state space and the ranges of parameter values. So, for practical purposes,

an easily implementable heuristic policy has been described. This heuristic performed favourably when compared to the optimal policy, as suggested by various experiments.

A natural generalization of the dynamic reconfiguration problem was to consider more than two job types and clusters. This led to a significantly more complex model, which was the object of study in chapter 3. Here, the optimal reconfiguration policy for a general number of server types, M, was now computed and tabulated. The state space increased dramatically in size as the number of job types increased. Hence, the computation was subjected to greater complexity constraints than those previously mentioned for the simpler model of two job types. It was now even more advantageous to develop a suitable heuristic which, although sub-optimal, could be calculated quickly and easily. The previous heuristic from section 2.5 was extended for use in this more general case. Once again this heuristic was shown to perform well in a range of experiments.

The validity of the mathematical models here described for dynamic server allocation, and in particular the heuristic outlined in section 3.5, has been demonstrated with the development of a prototype dynamic Resource Management System using Condor [44]. Initialised as part of the GridSHED project [33], this prototype is fully described in [17]. The prototype demonstrates the functionality of server and cluster *managers*, and shows the operation of the heuristic switching policy. The heuristic is used as a reconfiguration policy to make switching decisions, reallocating servers dynamically between pools of resources as demand changes. Experiments with real jobs (for example, binary programs such as simulations with asso-

ciated data and transactions) show that, for an identical workload pattern, response times are lower when the prototype reconfigures the compute resources dynamically compared to when they are not. This is encouraging, supporting the theoretical simulated results which predicted the benefits of dynamic reconfiguration outlined in section 3.4.

In chapter 4, the picture of the service provisioning system was broadened from the dynamic reconfiguration at the cluster level (where one cluster may offer different services, with the cluster resources being reconfigured dynamically if necessary) to consider a larger network of resources. Now under discussion was the underlying structure of these compute resources: how should the network be configured in order to optimize its performance? In section 4.2, two level tree structures of master nodes and dependents were considered. Through numerical analysis, a tree structure with a maximum of two levels was decided upon to minimize the overall average response time. Numerical results were obtained for this special case, which suggested that the benefits of dynamic reconfiguration of the network as load changes will indeed reduce long term average response times and help make the service provisioning system more commercially viable. A simple heuristic was described for the configuration of a network with a given number of service nodes and specified decision making speeds. This heuristic gave an adequate configuration which could be improved upon as the network load increases.

As network load increases further, a further tier of master nodes is necessary to avoid saturation. This led to the problem of considerably increased complexity which was discussed in section 4.3. Also, the model outlined in section 4.2 contained the assumption that the routing decision made by a

master node was to be one of random choice. This was not particularly realistic. Different models needed to be considered in more detail, and their effect on overall average response times noted. It was considered more likely that a master node would make an *intelligent* decision based on shortest queue or round-robin. This was discussed in detail in section 4.3. The extended model demonstrated that the topological structure of a service network has a significant effect on its performance. The optimal structure was obtained numerically for a given set of parameters, using existing analytical results applied to either random or cyclic routing policies. Searching through all possible configurations of a multi level tree structure is computationally expensive (certainly time consuming). Hence a simple heuristic was derived, extended from the heuristic for the simpler 2-level model from section 4.2. Once again, this heuristic was used to give a reasonable configuration for the network, which could be further improved as network load increases.

In chapter 5, the cost of providing service provisioning systems was once again analysed, from a different perspective. An attempt was made to improve the realism of models used to evaluate and optimize multi-server systems subject to breakdowns and repairs. A large, real-life data set was analysed statistically. This analysis showed that the operative periods of servers were not distributed exponentially, but instead a good fit was obtained with a hyperexponential distribution. Further analysis showed that the inoperative periods of the servers were more reasonably assumed to have an exponential distribution, although a hyperexponential distribution would be more accurate for them too.

A model with hyperexponentially distributed operative and inoperative

periods was formulated and solved exactly and approximately. The solution yielded performance metrics such as the average number of jobs present or, equivalently, the average response time. A number of numerical experiments were performed using these solutions. Various service provisioning problems concerning quality of service and cost were addressed. These included predicting the minimum number of servers that would be required to ensure a desired level of performance.

6.1 Future Work

Service provisioning systems such as the *Computing Grid* continue to be a popular research topic. In addressing the problems outlined here, an attempt has been made to develop a framework for strategies which should maximize efficiency in generic hosting environments.

Further work in the area of dynamic reconfiguration would be certainly be beneficial. The models presented in chapters 2 and 3 could be generalized to include service times which are non-exponentially distributed and arrival processes which are not Poisson. For example, requests for service may be prove to be bursty which would be more appropriately modelled by a Markov-modulated arrival process. Or, the application of the generalised exponential (GE) distribution [25] to service times may prove interesting.

The behaviour of the 'static' policy for dynamic switching outlined in section 3.5 could be further investigated. This policy allocates servers to different job types roughly in proportion to the offered load. However, as load increases or the number of servers increases, the performance of this

simple policy is such that the system quickly becomes saturated and associated holding costs are high. The reasons for this behaviour are as yet undetermined.

The results presented in chapter 4 outlined strategies for finding the optimal configuration of a large network of compute resources. A configuration close to optimal could be easily predicted from a simple heuristic. It was assumed that such reconfigurations would be performed by a system administrator. The administrator would have available to him all the parameters of the system, such as the average time for a routing decision to be made. In reality, one may wish to reconfigure the network dynamically, in response to changing demand. Here it was assumed that such reconfigurations are rare, but if they are not, then their cost should be taken into account. That would mean solving a much more complex dynamic optimization problem; it is a topic of future research. An investigation into why servers may be clustered together in the real-world to form trees of resources would be interesting: clustering may be appropriate either for geographically linked resources or instead for resources offering particular services. Another useful generalization would be to consider asymmetric configurations, with different parameters on different branches of the tree. That would not require different analytical tools, but would mean a big increase in the size of the search space.

The optimizations presented for dynamic server allocation and optimal tree structures in chapters 2, 3 and 4 could be combined for another interesting generalization. Consider a dynamically reconfigurable tree structure where each service cluster is itself dynamically reconfigurable. This again would lead to a much more complex dynamic optimization problem; even if

this proved to be intractable, heuristic policies may be able to minimize the overall response time of a widely distributed *Computing Grid.*

The optimizations presented in chapters 4 and 5 could potentially be applied to *peer-to-peer* (P2P) systems. P2P architecture describes a type of network in which each workstation or node has equivalent capabilities and responsibilities, computing power being offered from individual nodes throughout the entire network. Such systems may combine the idle or unused CPU processing power and/or free disk space of many nodes in the network. This is a simple architecture, but it is prone to poor performance under heavy loads. A challenge in P2P systems (see [26]) is optimizing the performance of the network as the system evolves while nodes are continuously joining and leaving. Many P2P systems build dynamic overlay networks that attempt to route requests efficiently to nodes that can satisfy them. These must adapt in the presence of changing loads and node failure. Therefore, it would be interesting to investigate whether the optimizations for network configuration presented in chapter 4 and the inclusion of possible breakdown patterns from chapter 5 could be used to improve the performance of P2P systems.

The analysis in chapter 5 of a real-life data set containing server breakdown and repair information, and the model of such a large system of servers yielded solutions which can be used to determine the distribution of the queue size. Hence the average queue size and the average response time may also be obtained. However, they do not provide the distribution (e.g., the 90% percentile) of the response time. Hence, an open problem in this area concerns the distribution of response times. Using real-life data sets for analysis, that would be an interesting topic for future research.

Bibliography

[1] J. Bather. *Decision Theory - An Introduction to Dynamic Programming and Sequential Decisions.* Wiley, 2000.

[2] D. Blackwell. Discounted dynamic programming. *Annals of Mathematical Statistics*, 26:226–235, 1965.

[3] J.P.C. Blanc. The power-series algorithm applied to the shortest-queue model. *Operations Research*, 40:157–167, 1992.

[4] C. Buyukkoc, P. Varaiya, and J. Walrand. The $c\mu$-rule revisited. *Advances in Applied Probability*, 17:237–238, 1985.

[5] R. Chakka and I. Mitrani. Heterogeneous multiprocessor systems with breakdowns: Performance and optimal repair strategies. *Theoretical Computer Science (Special Issue on Probabilistic Modelling)*, 125:91–109, 1994.

[6] J.S. Chase, D.E. Irwin, L.E. Grit, J.D. Moore, and S.E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *Proc. 12th IEEE International Symposium on High Performance Distributed Computing*, 2003.

[7] C. Clark et al. Live migration of virtual machines. In *Proc. 2nd Symposium on Networked Systems Design and Implementation*, 2005.

[8] G. Cybenko. Dynamic load balancing for distibuted memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[9] J.F. Dantzer, I. Mitrani, and Ph. Robert. Large scale and heavy traffic asymptotics for systems with unreliable servers. *Queueing Systems*, 38:5–24, 2001.

[10] E. de Souza e Silva and H.R. Gail. *Performability Modelling: Techniques and Tools*, chapter The Uniformization Method in Performability Analysis. Wiley, 2001.

[11] M. Dobber, G. Koole, and R. van der Mei. Dynamic load balancing for a grid application. In *Proc. HiCP 2004*, pages 342–352. Springer LNCS, 2004.

[12] M. Dobber, G. Koole, and R. van der Mei. Dynamic load balancing experiments in a grid. In *Proc. CCGrid 2005*, 2005.

[13] S.E. Dreyfus and A.M. Law. *The Art and Theory of Dynamic Programming*. Academic Press, 1977.

[14] I. Duenyas and M.P. Van Oyen. Stochastic scheduling of parallel queues with set-up costs. *Queueing Systems Theory and Applications*, 19:421–444, 1995.

[15] I. Duenyas and M.P. Van Oyen. Heuristic scheduling of parallel heterogeneous queues with set-ups. *Management Science*, 42:814–829, 1996.

[16] G. Fayolle, P.J.B. King, and I. Mitrani. The solution of certain two-dimensional markov models. *Advances in Applied Probability*, 14:295–308, 1982.

[17] M. Fisher, C. Kubicek, P. McKee, I. Mitrani, J. Palmer, and R. Smith. Dynamic allocation of servers in a grid hosting environment. In R. Buyya, editor, *Proc. Fifth IEEE/ACM International Workshop on Grid Computing (Grid 04)*, pages 421–426, Pittsburgh, USA, 2004. IEEE Computer Society.

[18] J. Hine and T. Holzer. Task allocation in a distributed system. In *Proc. UniForum'97: Untangling the Web*, 1997.

[19] M. Hofri and K.W. Ross. On the optimal control of two queues with server set-up times and its analysis. *SIAM Journal of Computing*, 16:399–420, 1987.

[20] M. Houle, A. Symvonis, and D. Wood. Dimension-exchange algorithms for load balancing on trees. In *Proc. 9th Int.Colloquium on Structural Information and Communication Complexity*, pages 181–196, 2002.

[21] F.J. Massey Jr. The kolmogorov-smirnov test of goodness of fit. *Journal of the American Statistical Association*, 46:68–78, 1951.

[22] L. Kleinrock. *Queueing Systems Volume 1: Theory*. Wiley. 1975.

[23] G. Koole. Assigning a single server to inhomogeneous queues with switching costs. *Theoretical Computer Science*, 182:203–216, 1997.

[24] G. Koole. Structural results for the control of queueing systems using event-based dynamic programming. *Queueing Systems Theory and Applications*, 30:323–339, 1998.

[25] D.D. Kouvatsos. Entropy maximisation and queueing network models. *Annals of Operations Research*, 48:63–126, 1994.

[26] D. Liben-Nowell et al. Analysis of the evolution of peer-to-peer systems. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 233–242. ACM Press, 2002.

[27] W. Lin and Kumar P. R. Optimal control of a queuing system with two heterogeneous servers. *IEEE Transactions on Automatic Control*, AC-29, 1984.

[28] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proc. 8th International Conference of Distributed Computing Systems*, pages 104–111, 1988.

[29] Z. Liu, P. Nain, and D. Towsley. On optimal polling policies. *Queueing Systems Theory and Applications*, 11:59–83, 1992.

[30] E. Manoel et al. Provisioning on demand: Introducing IBM Tivoli Intelligent ThinkDynamic Orchestrator. http://www.redbooks.ibm.com, 2003.

[31] Sun Microsystems. N1 grid - introducing just in time computing. http://wwws.sun.com/software/solutions/n1/wp-n1.pdf, 2003.

[32] I. Mitrani. *Probabilistic Modelling.* Cambridge University Press, 1998.

[33] I. Mitrani. Research proposal: e-science solutions for grid scheduling, hosting and environment design, 2002. School of Computing Science, University of Newcastle-upon-Tyne.

[34] I. Mitrani. Approximate solutions for heavily loaded markov-modulated queues. *Performance Evaluation*, 62:117–131, 2005.

[35] I. Mitrani and B. Avi-Itzhak. A many-server queue with service interruptions. *Operations Research*, 16:628–638, 1968.

[36] I. Mitrani and B. Avi-Itzhak. A many-server queue with service interruptions. *Operations Research*, 16:628–638, 1968.

[37] I. Mitrani and R. Chakka. Spectral expansion solution for a class of markov models: Application and comparison with the matrix-geometric method. *Performance Evaluation*, 23:241–260, 1995.

[38] M.F. Neuts and D.M. Lucantoni. A markovian queue with n servers subject to breakdowns and repairs. *Management Science*, 25:849–861, 1979.

[39] J.M. Ortega and W.C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, 1970.

[40] J. Palmer and I. Mitrani. Optimal server allocation in reconfigurable clusters with multiple job types. In *Proc. Computational Science and its*

Applications (ICCSA 2004), volume 3044 of *Lecture Notes in Computer Science*, pages 76–86. Springer, 2004.

[41] J. Palmer and I. Mitrani. Optimal and heuristic policies for dynamic server allocation. *J. Parallel Distrib. Comput.*, 65:1204–1211, 2005.

[42] J. Palmer and I. Mitrani. Optimal tree structures for large service networks. In *Proc. 1st EuroNGI Conference on Next Generation Internet (NGI 2005)*, Rome, Italy, 2005. IEEE Computer Society.

[43] S. M. Ross. *Introduction to Stochastic Dynamic Programming.* Academic Press, 1983.

[44] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. *Grid Computing: Making The Global Infrastructure a Reality*, 2003.

[45] N. Ya. Vilenkin. *Combinatorics.* Academic Press, 1971.

[46] B. Vinod and T. Altiok. Approximating unreliable queueing networks under the assumption of exponentiality. *J. Oper. Res. Soc.*, 37:309–316, 1986.

[47] P. Whittle. *Optimisation over Time: Dynamic Programming and Stochastic Control*, volume 1,2. Wiley, 1982.

[48] M.J. Zaki, W. Li, and S. Parthasarathy. Customized dynamic load balancing for a network of workstations. In *Proc. 5th IEEE Int. Symp.*, pages 282–291. HDPC, 1996.