Routing and Transfers Amongst Parallel Queues

Thesis by

Simon P. Martin

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

Newcastle University

Newcastle upon Tyne, UK

2008

(Submitted March 16, 2008)

Abstract

This thesis is concerned with maximizing the performance of policies for routing and transferring jobs in systems of heterogeneous servers. The tools used are probabilistic modelling, optimization and simulation.

First, a system is studied where incoming jobs are allocated to the queue belonging to one of a number of servers, each of which goes through alternating periods of being *operative* and *inoperative*. The objective is to evaluate and optimize performance and cost metrics. Jobs incur costs for the amount of time that they spend in a queue, before commencing service. The optimal routing policy for incoming jobs is obtained by solving numerical programming equations. A number of heuristic policies are compared against the optimal, and one dynamic routing policy is shown to perform well over a large range of parameters.

Next, the problem of how best to deal with the transfer of jobs is considered. Jobs arrive externally into the queue attached to one of a number of servers, and on arrival are assigned a time-out period. Jobs whose time-out period expires before it commences service is instantaneously transferred to the end another queue, based on a routing policy. Upon transfer, a transfer cost is incurred. An approximation to the optimal routing policy is computed, and compared with a number of heuristic policies. One heuristic policy is found to perform well over a large range of parameters.

The last model considered is the case where incoming jobs are allocated to the queue attached to one of a number of servers, each of which goes through periods of being *operative* and *inoperative*. Additionally, each job is assigned a time-out on arrival into a queue. Any job whose time-out period expires before it commences service is instantaneously transferred to the end of another queue, based on a transfer policy. The objective is to evaluate and optimize performance and cost metrics. Jobs incur costs for the amount of time that they spend in a queue, before commencing service, and additionally incur a cost for each transfer they experience. A number of heuristic transfer policies are evaluated and one heuristic which performs for a wide range of parameters is observed.

Declaration

All work contained within this thesis represents the original work of the author. Most of the work in this thesis has been published in conference proceedings as detailed below. The material in chapters 2, Chapter 3 has been published in 1 and 2; some of the material in chapters 4 and 5 has been published in 3 and the material in chapter 6 has been published in 4, and is to be published in 5.

1. **Dynamic Routing Between Two Queues with Unreliable Servers**, *S. Martin and I. Mitrani, I.* International Journal of Simulation, Volume 5, Issue 5, pp 38-48 U.K. Simulation Society, 2004
2. **Dynamic Routing Among Several Intermittently Available Servers**, *S.P. Martin, I. Mitrani and K.D. Glazebrook*, Proceedings of the first EuroNGI Conference on Traffic Engineering for the next generation Internet (EURONGI 2005), Roma, Itlia, 18-20 April 2005
3. **Performance optimization with deadlines and job transfers**, *S. Martin and I. Mitrani*, Int. Conf. on Heterogeneous Networks (HET-NET '05), Ilkley 2005.
4. **Job Transfers Between Queues with Unreliable Servers**, *S. Martin and I. Mitrani* Proceedings of MCQT'06, Madrid, 2006
5. **Job Transfers Between Queues with Unreliable Servers**, *S. Martin and I. Mitrani*, Annals of Operations Research *(to be published)*

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The motivation for this work is recent developments in distributed computing. Advances in high performance hardware, and in particular the widespread availability of high speed networking, has led to the connection of previously isolated computing resources. This allows users to access remote servers with storage and processing capabilities which may be located anywhere within a local or wider area network.

The concept of a *Computing Grid* is defined as the technology enabling the coupling of such resources which may be both geographically and administratively dispersed. It is a desirable feature of such a system that the user does not need to know where or how their desired service is performed, but rather submitting a request for service to the Grid, and awaiting the results of the service. A computing grid therefore supports the concept of heterogeneous servers providing service to a widely distributed community of users. The role of the Grid management system is to maximise the efficiency of the composite servers. An environment can be considered more *efficient* than another if it makes a better use of all available resources. This can be analysed using performance measures such as the average number of waiting jobs present in the system or, following the submission of a job, the average response time. The latter is defined as the amount of time from the arrival of the service request until the completion of service.

An example of a *routing* system is illustrated in Figure 1.1. Requests for service

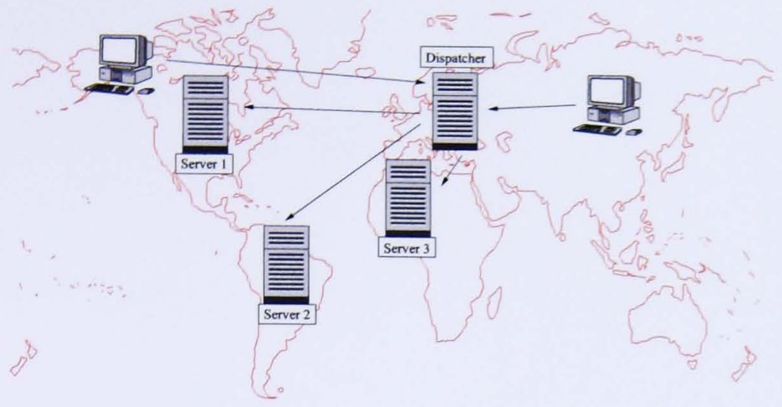

Figure 1.1: A routing-based service provisioning system

arrive into a provisioning system, or *dispatcher*, which is responsible for the allocation of service requests, or *jobs*, to available computing resources, for example the selection of a server for the job. The provisioning system may also be responsible for tracking the number of jobs queueing at each resource.

On the arrival of a request for service, the dispatcher routes the job to available computing resources, according to a routing policy. The routing policy determines the appropriate allocation of resources, based on service capacities and the current state of the resources, including the number of queueing jobs and the current resource availability.

For the purposes of this thesis, a dynamic routing policy is one which takes into account the current operational state of the resources associated with the dispatcher, and a static policy is one which does not. It is sensible to use dynamic routing when the queue sizes and server states are known centrally, and the costs of centrally dispatching jobs from the dispatcher to the destination server are low.

A different example of a *transfer* system is illustrated in Figure 1.2. Requests for service are submitted by the user to one of the available computing resources. The

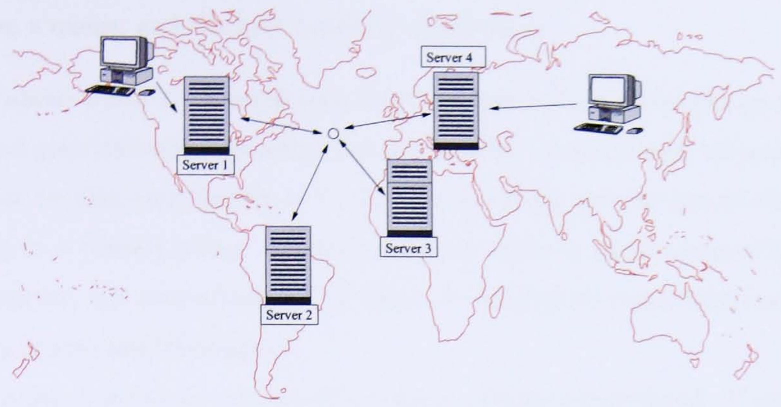

Figure 1.2: A transfer-based service provisioning system

job then enters a queue awaiting either service, or transfer to another resource, where it is expected to receive better service.

Job transfers make sense when either the system state information is not available or when sending jobs to remote servers is expensive.

It is easy to see that in the case of the routing-based system, for best utilization of resources, the provisioning system should take into account the current state of the queues at each resource in addition to the, potentially changing, performance characteristics of each.

In the case of transfer systems it is clear that jobs may be transferred on arrival to another resource if there is a clear benefit to doing so. In many cases, however, it may not be certain whether the best option for a job is to remain in a queue, rather than transferring to another, possibly distant, queue. This decision will be made more complex if the performance characteristics of resources vary with time.

Objectives of this thesis include:

1. to determine policies for a routing-based provision system to select the destination resource for an incoming job;

2. to determine heuristics for transfer systems, to determine when to transfer jobs

from a queue, and the best resulting destination;

In Chapters 2 and 3, a routing system is considered. Each server has an associated queue, and goes through alternating periods of being *operative* and *inoperative*. The role of the provisioning system is to allocate incoming jobs to one of the servers, according to a *routing policy*. A static policy is likely to cause under-utilisation of some resources, and over-utilisation of others. Strategies for *semi-static* and *dynamic* allocation of jobs are investigated.

In Chapter 4, a simple example of a transfer system is considered. There are only two servers, with transfers permitted in only one direction. The effect of different transfer rates is evaluated, and the system solved exactly for a performance metric. A good approximation to the exact solution is obtained and evaluated.

In Chapters 5 and 6 general transfer systems are evaluated, firstly for always available servers, and then for servers which undergo alternating periods of being operative and inoperative. A number of transfer policies are evaluated.

1.2 Related Work

1.2.1 Routing

Foley and McDonald [9] consider a system of m servers where each server has a *dedicated* stream of customers in addition to a shared stream of *smart* customers which use the shortest queue routing policy. They describe three separate cases where one or both servers can overload, dependent on the server speeds and the proportion of *smart* customers.

The stability conditions are computed for the general case of m servers. Additionally, the exact asymptotic distribution is computed for the limited case of $m = 2$.

1.2.2 Server Breakdowns

Thomas and Mitrani [22] consider N parallel queues which undergo independent operative and inoperative periods, which are all supplied with jobs from a single incoming stream. Incoming jobs are routed to one of the queues based on a routing policy which only depends on the set of operative states of all servers. This is a static version of the system considered in Chapters 2 and 3. Several routing strategies are evaluated and compared with each other and the optimal static routing policy.

Mitrani and Wright [16] consider a system where incoming jobs form a single incoming stream which can be routed through N parallel M/M/1 queues. These servers are subject to random breakdowns and repairs. When a server breaks down, all jobs present in its corresponding queue are lost. Servers which are broken down when jobs arrive into the system cannot receive jobs into their queues.

The marginal queue size distributions are computed for the general case, and in the special case of $N = 2$, the equilibrium distribution of the numbers of jobs in the queues is calculated.

Wang, Wang and Pearn [23] study a single, unreliable server in the N policy M/G/1 queueing system with startup times. When N customers are waiting the server starts a 'warm-up' period, where it is unable to commence service. Thereafter, the server provides service until the queue becomes empty, or it undergoes a breakdown.

Approximate formulas are derived for the steady-state probability distributions of the queue length, using the maximum entropy principle, and compared against established results for various distributions. It is shown that the maximum entropy approach generates a good approximation and hence is a useful approach.

Gray, Wang and Scott [11] consider a queueing model in which a single server may experience several different types of breakdowns, each of which requires a random, finite number of stages of repair. Necessary conditions for the existence of a stationary queue length distribution to occur is obtained, and then matrix geometric methods are used to compute the queue length distribution, and then an explicit expression for its mean.

A number of properties of the model are also found, including the mean repair time, average completion time, average number of repair stages and relationships between these measures and the number of breakdowns during a customer's service time.

Glazebrook and Kirkbride [10] consider a model in which service times and repair times at each of number of machines are independent and identically distributed random variables with general distribution. Routing decisions take into account queue lengths, machine operative status and elapsed processing time of jobs in service.

An approach to machine calibration is developed which gives a machine index which is a function of all status information. A number of heuristics are developed and compared against the optimal and one heuristic, consisting of routing tasks to the machine with the smallest current index, is identified which performs very well. It is noted that the approach is very flexible and will yield good policies for a range of variants of the basic model.

1.2.3 Reneging

Zeltyn and Mandelbaum [28] consider the problem of a single queue with Poisson arrivals, with n statistically-identical agents which service the queue. Each arriving caller has an associated, generally distributed *patience time* τ. The model is applied to call center environments.

Three asymptotic operational regimes for the number of agents are studied, and one is found to be a good approximation for a wide set of system parameters. In addition, parameters where simpler approximations are useful are identified.

Altman and Yechiali [1] investigate a series of models considering customer impatience, which is due to the *absence* of service on arrival. When a customer arrives into a queue and notices that the server is 'on vacation', an impatience counter is started. If the server returns to service before the timer expires, the customer remains in the system until its service is completed. Alternately, if the timer expires before the server returns to service, the customer leaves the system never to return.

Single server M/M/1 and M/G/1 queues are analysed, as is the multi-server M/M/c queue, and closed-form results are obtained. In particular, the proportion of customer abandonments is calculated and compared for single-vacation and multi-vacation regimes, and it is found that this is smaller for the single-vacation case.

Dalal and Jordan [7] consider a M/M/1 queue in which the average reward for servicing a job is a decreasing function of the sojourn time. The maximum reward and mean service times of a job are independent, arbitrarily distributed, random variables. Deadlines are not known by the server, and hence expired jobs are not dropped. A scheduler, which is assumed to know the maximum reward, service rate and age of each job, selects the next job to receive service on the completion of the previous job.

Various schedulers are compared by simulation over a range of loads. It is proved that a scheduling policy that serves the customer with the highest product of *potential reward* and service rate, maximizes the average reward.

He and Neuts [12] study a system of two servers with job transfers. Batches of jobs are transferred from the longer queue to the shorter one when the difference between them reaches a threshold L. The arrival rates and service rates of each server are independent.

A simple condition for the stability of the system is obtained. A matrix geometric solution for the stationary distribution of the system is then calculated, and then the stationary distribution of the total number of jobs in the system is obtained and shown to decay exponentially. Based on theoretical results, the optimal system parameters of such queueing systems are explored numerically.

Movaghar [17] studies queueing systems where customer have strict deadlines until the beginning of service. A single queue serves a number of identical servers, and may have a finite capacity. An arriving job which finds the system full leaves immediately, never to return.

Equations for probability density function of the time an infinite deadline customer has to wait until service commences are calculated for both finite and infinite capacity systems, as well as the probability of missing deadlines, and the probability of blocking. The efficacy of the method is illustrated numerically.

Choi, Kim and Zhu [6] consider a MAP/M/c queue where a customer which cannot commence within a fixed time after arrival into the system is lost. The queue is serviced by c servers. Two cases of clients are considered: "aware" and "unaware" customers. In the case of aware customers, an arriving customer knows how long they can expect to wait, and waits in the queue or leaves the system on arrival, depending whether they will start service before the deadline expires. In the case of unaware customers, the customer always enters the system and is lost if service does not start before the deadline expires.

The stationary distribution of the system is obtained and hence several performance measures such as loss probability, waiting time distribution, mean waiting time and mean queue size are computed. Numerical examples are presented for system load of 0.7, 0.8 and 0.9, with differences presented, where they exist, for aware and unaware customers.

Liu and Kulkarni [13] evaluate a system with balking based on the workload. On arrival into the system, a job enters the system, and remains until service completion, if the expected waiting time is less than some maximum. Otherwise, it leaves the system, never to return.

Some results are obtained for the case of a M/G/1 queue, and then the case of a M/PH/1 queue is solved explicitly, to produce the probability that the system is empty, and the mean workload in equilibrium. A number of numerical examples are presented.

Bae, Kim and Lee [3] study the M/G/1 queue with impatient customers, which leave the system if, after a fixed time K, they have not started service. It is noted that when analysing the waiting time and busy period, that this is equivalent to the case where customers only enter the system when their waiting time does not exceed K.

The distribution of the waiting time is explicitly derived, and the expected busy period is obtained.

Skimkin and Mandelbaum [20] consider the modelling of abandonment from an M/M/m queue, where each customer has a deadline, T, after which they will abandon the system if they have not commenced service. The customers are placed into a number of types, based on three utility function parameters.

The optimal (or rational) behaviour for customers, which maximises their utility function is conpared to a *myopic decision rule* which chooses the abandonment time as the first local maximum of the utility function. This is shown to enjoy favourable analytical properties, in addition to making sense for cases where customers do not know the exact form of their utility functions. Concrete examples are provided, which illustrate the approach and analysis.

Ward and Glynn [24] consider a GI/GI/1 queue with customers which which either balk or reneging. In the reneging case, customers leave the queue if they have not commenced service before a deadline expires. For the case of customer balking, on arrival a customer does not enter the system if the offered waiting time exceeds the deadline, for the case where all customer processing times are known, or the conditional expected waiting time when only queues lengths are observable.

An approximation for the workload and queue length processes is obtained, for the case where the arrival rate is close to the processing rate with large reneging times, and this is shown to be a good approximation. Their approach is also shown to generate a good approximation for the queue-length process.

Zhao and Grassman [29] solve a shortest queue problem, in which arriving jobs are always routed to the shortest queue, and transfers of jobs between queues are permitted. The servers are always available, and jobs transfer instantaneously between two queues. When the difference in queue length between the longest and shortest queue exceeds a pre-set number, the last job is transferred to the shortest queue.

Expressions of main performance measures, including the average number of jobs in the system, the average waiting time in the system and the average number of transfers, are given. A number of numerical results are presented, and by comparing the results for systems with and without jobs transfers, it is shown that a significant improvement of the system performance is achieved for the system with job transfers.

Xu and Zhao [27] consider a system where incoming jobs are routed to one of two servers. Transfers of jobs are permitted between the two servers in either direction. A transfer cost is incurred whenever a job is transferred.

Dynamic routing and transfer policies are characterised that minimise the expected total cost for both discounted and long-run average costs. It is shown that the optimal routing and transfer controls are described by three monotonically non-decreasing functions. Properties of these functions, relationships between them and their asymptotic behaviour are considered, and it shown that some well-known queueing control models are special cases of their model.

Boots and Tijms [5] examine a multiserver queueing system with impatient customers. Each arriving customer is placed into a shared queue, and has an associated time-out. If the customer has waiting for their time-out period, and has not begun service, they leave system and become a lost customer. The loss probability, which is the long-run fraction of customers who are lost, is formulated for the M/M/c queue, and an approximation for the M/G/c queue is formulated and evaluated against numerical results, showing it to be a good approximation.

Chapter 2

Servers Subject to Breakdowns and Repairs: Optimal Routing

2.1 Introduction

Grid services involve groups of heterogeneous servers, providing service to widely distributed users. These users submit jobs without necessarily knowing or caring on which server they will be executed. It is the responsibility of the system to allocate these jobs among the servers, attempting to make the best use of available resources, and provide the best quality of service.

In many implementations, grid services are run with a lower priority than other services, such as print or web services, on a server. This can be modelled by each server experiencing periods of availability, separated by unavailable periods. The problem then is to select, for a given operative state and queue length for each server, the optimal routing decision for arriving jobs.

This can, in principle, be solved exactly to yield the optimal routing policy. However, for large numbers of servers, this is would take an unfeasible length of time to calculate, and a very large amount of storage. Therefore, the optimal policy will be calculated for a small number of servers, where the computation time and storage requirements are not overly large, and compared with several heuristic policies. Then, for larger numbers of servers, comparison between the heuristic policies will be made.

2.2 The model

Jobs arrive into the system according to an independent Poisson process with rate λ. A routing policy sends the new arrivals to one of N servers, each having its own unbounded FIFO queue. There is no delay between arriving into the system and joining a queue. Having joined, a job remains in its queue until its service is completed. When server i is operative, its service times are distributed exponentially with mean $1/\mu_i$ $(i = 1, 2, \ldots, N)$. The operative and inoperative periods of server i are distributed exponentially with means $1/\xi_i$ and $1/\eta_i$, respectively. Any job whose service is interrupted by a server breakdown remains at the head of its queue; as soon as the server is repaired, the service resumes from the point of interruption. All interarrival, service, operative and inoperative intervals are mutually independent. This system is illustrated in Figure 2.1.

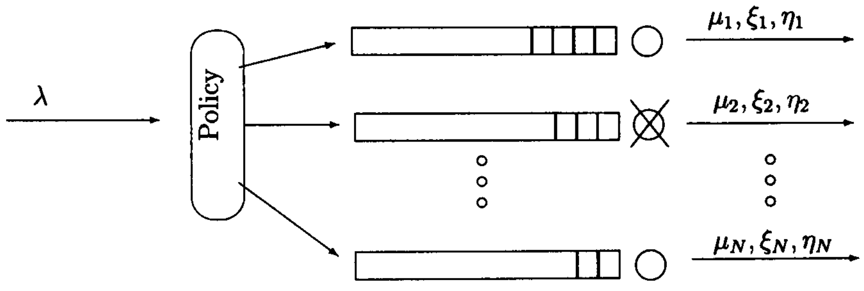

Figure 2.1: Unreliable Servers

While a job remains in queue i, it incurs a cost c_i per unit time. These 'holding' costs reflect the possibly different importance attached to low response times at the N queues. The average total cost incurred over a given (finite or infinite) period will be our QoS measure.

The system state, S, at a given time is described by a vector of integers:

$$S = (j_1, j_2, \ldots, j_N, b_1, b_2, \ldots, b_N) \ ,$$

where j_i is the current number of jobs in queue i, and b_i is the current availability of server i ($i = 1, 2, \ldots, N$); the latter is defined as

$$b_i = \begin{cases} 0 & \text{if server } i \text{ is inoperative} \\ 1 & \text{if server } i \text{ is operative} \end{cases} .$$

The routing policy, u, is defined by specifying, for every state S, the action, u_S, taken when a job arrives and finds that state: $u_S = i$ if the job is directed to queue i. The policy is assumed to be stationary; the routing actions may depend on the current state but not on past history.

The above assumptions imply that the system state is a Markov process whose evolution depends on the routing policy. The instantaneous transition rate, $r_u(S, S')$, from state S to state S' under policy u, is given by

$$r_u(S, S') = \begin{cases} \lambda & \text{if } u_S = i \text{ and } S' = S + e_i; \\ \mu_i b_i & \text{if } j_i > 0 \text{ and } S' = S - e_i; \\ \xi_i & \text{if } b_i = 1 \text{ and } S' = S - e_{i+N}; \\ \eta_i & \text{if } b_i = 0 \text{ and } S' = S + e_{i+N}; \\ 0 & \text{otherwise} \end{cases} , \tag{2.1}$$

where $i = 1, 2, \ldots, N$; e_k is the 2N-dimensional vector whose kth element is 1 and the other $2N - 1$ are 0.

The total instantaneous transition rate out of state S, $r(S)$, is equal to:

$$r(S) = \lambda + \sum_{i=1}^{N} (b_i[\mu_i \delta(j_i > 0) + \xi_i] + (1 - b_i)\eta_i) \ , \tag{2.2}$$

where $\delta(B)$ is the indicator of the Boolean B: it is equal to 1 if B is true, 0 if B is false. Note that $r(S)$ does not depend on the routing policy.

If the routing policy makes reasonably efficient use of the servers, i.e. does not allow one of the queues to grow very large while others remain empty, then the system should be stable if the arrival rate is lower than the total average available service capacity:

$$\lambda < \sum_{i=1}^{N} \left(\frac{\eta_i}{\xi_i + \eta_i} \mu_i \right) . \tag{2.3}$$

2.3 Computation of the optimal policy

The optimization problem consists of finding the minimal cost, C^{min}, and a stationary routing policy that achieves it:

$$C^{min} = \inf_u \sum_{i=1}^{N} c_i L_{u,i} , \tag{2.4}$$

where $L_{u,i}$ is the mean queue length of server i under policy u, and c_i is the holding cost per job per unit time for server i as before.

For the purposes of optimization, it is convenient to apply the technique of uniformization to the Markov process (e.g., see [8]). This involves the introduction of 'fictitious' transitions which do not change the system state, in such a way that the average interval between consecutive transitions does not depend on the state. The discrete-time Markov chain embedded at transition instants is then equivalent to the original process. First, we find a constant, Λ, such that $r(S) \leq \Lambda$ for all S. A suitable value for Λ is

$$\Lambda = \lambda + \sum_{i=1}^{N} \max(\mu_i + \xi_i, \eta_i) . \tag{2.5}$$

Without loss of generality, the unit of time can be scaled so that the right-hand side of (2.5) is equal to 1. Then the transitions of the Markov process can be assumed to occur at exponentially distributed intervals with mean 1, according to a discrete-time Markov chain whose one-step transition probabilities under policy u, $q_u(S,S')$, are equal to

$$q_u(S,S') = \begin{cases} r_u(S,S')/\Lambda & \text{if } S' \neq S \\ 1 - r(S)/\Lambda & \text{if } S' = S \end{cases} , \tag{2.6}$$

with $r_u(S,S')$ and $r(S)$ given by (2.1) and (2.2) respectively.

For the purpose of accumulating costs, we consider the states of the above Markov chain *just after* a transition instant if the latter is not associated with an arrival and *just before* if an arrival occurs at that instant. Thus, if the chain is in state S and there is no arrival, then the cost of the current step, $v_0(S)$, is equal to

$$v_0(S) = \sum_{i=1}^{N} c_i j_i , \tag{2.7}$$

taking the necessary adjustments to c_i due to adjustment in the unit of time as given.

If the state is S and an arrival occurs, then in addition to $v_0(S)$, a holding cost equal to c_i if $u_S = i$ is incurred.

Suppose that the objective is to minimize the average total cost incurred over a finite period consisting of n steps of the Markov chain. Denote by $V_n(S)$ the minimum of that average, given that the current state is S and there is no arrival. Similarly, let $V_n^a(S)$ be the minimum average total cost, given that the current state is S and an arrival occurs (with the a subscript denoting the arrival). These costs satisfy a set of dynamic programming equations (for the general theory, see [19, 25]).

If there is no arrival in the current state, we have

$$V_n(S) = v_0(S) + \lambda V_{n-1}^a(S) + \sum_{S'} q_u(S,S')V_{n-1}(S') , \tag{2.8}$$

where the first term in the right-hand side is the cost of the current step. The second term expresses the fact that the next transition is an arrival with probability λ; if so, the incoming job sees state S and the consequent cost of the remaining $n-1$ steps is $V^a_{n-1}(S)$. The sum in the third term extends over the transitions $S \to S'$ which do not involve an arrival: the next state is S' with probability $q_u(S, S')$; if so, the consequent cost of the remaining $n-1$ steps is $V_{n-1}(S')$.

When there is an arrival in the current state, one of the routing actions directing the incoming job to queue i must be taken ($i = 1, 2, \ldots, N$). The state then immediately jumps to $S + e_i$. The cost $V^a_n(S)$ is therefore obtained by adding to the current holding cost, $v_0(S)$, the minimum of the consequences of this action (on the current and subsequent $n-1$ steps), over the possible actions:

$$V^a_n(S) = v_0(S) + \min_{i=1,2,\ldots,N}\left[c_i + \lambda V^a_{n-1}(S+e_i) + \sum_{S'} q_u(S+e_i, S')V_{n-1}(S')\right] . \quad (2.9)$$

Again, the sum in the right-hand side extends over the transitions $S + e_i \to S'$ which do not involve an arrival.

The above recurrences can, in principle, be solved by iteration, starting with the initial values $V_0(S) = v_0(S)$ and $V^a_0(S) = v_0(S) + \min(c_1, c_2, \ldots, c_N)$. In practice, the state space must be made finite by bounding the queue sizes: $j_i \le J_i$ and for some J_i, where $i = 1, 2, \ldots, N$. The consequences of such a truncation are that incoming jobs are only routed to one of the queues for which $j_i < J_i$; if $j_i = J_i \forall i \in \{1, 2, \ldots, N\}$, new arrivals are lost.

The complexity of the iterative solution is of the order $O(n \prod_{i=1}^{N}(2J_i))$, since the size of the state space is $\prod_{i=1}^{N}(2J_i)$ and there are n steps (the summations in (2.8) and (2.9) have no more than 4 terms each).

Having solved the equations, the value of i which achieves the minimum in the right-hand side of (2.9) is the optimal routing action in state S, for the finite horizon n.

More commonly, one is interested in an infinite-horizon optimization. The objective is to minimize the average total future cost, and in order that the latter is finite, the cost of a step at distance n in the future is discounted by a factor α^n, for some $0 \leq \alpha < 1$. Setting $\alpha = 0$ implies that all future costs are disregarded; only the current step is important. When $\alpha \to 1$, the weight of a future step, no matter how distant, approaches that of the current one.

Dynamic programming is a mathematical optimisation technique, which allows the total cost of a decision to be minimised, including future consequences. Decisions cannot be viewed in isolation since one must balance the desire for low present cost with the undesirability of high future costs. The dynamic programming technique captures this trade-off. At each stage, decisions are ranked based on the sum of the present cost and the expected future cost, assuming optimal decision making for subsequent stages [4].

Denote by $V(S)$ the minimum average total future cost, given that the current state is S and there is no arrival. Similarly, $V^a(S)$ is the minimum average total future cost, given that the current state is S and an arrival occurs. The corresponding dynamic programming equations are

$$V(S) = v_0(S) + \alpha\lambda V^a(S) + \alpha \sum_{S'} q(S, S')V(S') \, , \tag{2.10}$$

$$V^a(S) = v_0(S) + \min_{i=1,2} \left[c_i + \alpha\lambda V^a(S + e_i) + \alpha \sum_{S'} q(S + e_i, S')V(S') \right] \, , \tag{2.11}$$

with the same restrictions on S' as for (2.8) and (2.9), respectively.

Again, the optimal routing action in state S is specified by the value of i which achieves the minimum in the right-hand side of (2.11).

The infinite horizon optimization leads to fixed-point equations, rather than recurrent ones. Moreover, their solution, and the optimal policy, depend on the discount factor α. The most important case, but also the most difficult to solve, is $\alpha \to 1$. Three methods for computing the optimal policy numerically are described below. In all cases, the state space is truncated by introducing the bounds $j_i \leq J_i$, and imposing the appropriate policy restrictions on the boundaries $j_i = J_i$ $(i = 1, 2, \dots, N)$(see above).

Cost Iteration

This algorithm applies when $\alpha < 1$. Then (a) the total costs are finite and (b) the finite horizon costs and policy converge to the infinite horizon costs and policy as $n \to \infty$. The algorithm works as follows:

1. At iteration 0, set $V_0(S)$ to $v_0(S)$ and $V_0^a(S)$ to $v_0(S) + min(c_1, c_2, \dots, c_N)$, for all states S.

2. At iteration n, compute $V_n(S)$ and $V_n^a(S)$ according to (2.8) and (2.9) respectively, using $V_{n-1}(S)$ and $V_{n-1}^a(S)$ from iteration $n-1$. Terminate when

$$\max_S [|V_n^a(S) - V_{n-1}^a(S)|] < \epsilon \,, \tag{2.12}$$

   for some small ϵ.

3. Return the policy specified by the values of i which achieve the minima in the right-hand side of (2.9) on the last iteration.

The complexity of the *Cost Iteration* algorithm is of the order $O(n \prod_{i=1}^{N} J_i)$, where n is the number of iteration steps needed for convergence. That number depends on the model parameters, on the discount factor, α, and on the desired accuracy, ϵ.

Policy Stability

This is similar to cost iteration, but is applied with $\alpha = 1$. Now $V_n(S)$ and $V_n^a(S)$ keep growing without bound, so a different termination criterion must be used. This is based on convergence of policy, rather than convergence of cost. At each iteration, the current 'optimal' policy is compared to the one from the previous iteration. The algorithm terminates if the policy has not changed for k consecutive iterations, for some k (e.g., $k = 100$). Return that policy.

The complexity of this algorithm is of the order $O(n \prod_{i=1}^{N} J_i)$, where n is the number of iteration steps needed to achieve policy stability. That number depends on the model parameters and on the desired degree of stability, k.

Policy Improvement

Like cost iteration, this algorithm applies when the total costs are finite ($\alpha < 1$). However, it iterates on policy rather than costs, and ensures that the optimal policy is found.

1. Start by making an initial guess about the optimal policy, i.e. construct an initial mapping, $f(S)$, from system states to routing actions. This could be a simple heuristic such as $f(S) = i$ if $j_i = \min_{i'} j_{i'}$, (i.e. send new arrivals to the shorter queue).

2. Treat this guess as the optimal stationary policy and write the corresponding discounted cost equations. The only change with respect to (2.10) and (2.11) is that in the right-hand side of (2.11) there is no $\min_i$; the routing action $f(S)$ is used. This new version of (2.10) and (2.11) is a set of simultaneous linear equations for $V(S)$ and $V^a(S)$. Solve them and determine the costs associated with policy f.

3. Now try to 'improve' policy f. For every state S, find the routing action i

which achieves the minimum value in the original equation (2.11). In other words, minimize the total cost in state S, assuming that *after the current step*, policy f will be used.

4. If the new routing actions are the same as $f(S)$ for all S, then the policy f cannot be improved; it is optimal. Return f. Otherwise, replace $f(S)$ by the new policy and repeat from step 2.

In step 2, the simultaneous set of linear equations is very sparse and is normally solved by iterations. Therefore, the complexity of the Policy Improvement algorithm is of the order $O(mn\prod_{i=1}^{N}(J_i))$, where m is the number of steps in the iterative solution of the simultaneous equations, and n is the number of policy improvement steps. The number m depends on the model parameters, on the discount factor, and on the desired accuracy; in addition, n depends on how close the initial guess is to the optimal policy.

Of the above three algorithms, only *Policy Improvement* is guaranteed to produce the optimal routing policy in a finite number of steps (assuming that the simultaneous equations are solved accurately).

Optimal routing policies can, in principle, be computed off-line and stored in the form of decision tables. A dispatcher could then implement the policy by means of table look-up. A part of such a decision table is illustrated in Table 2.1. For each state where the queue sizes are in the range 0-18, server 1 is operative and server 2 is inoperative, the table indicates whether an incoming job should be sent to queue 1 or to queue 2. There are similar tables for the other three operative states (broken-operative, operative-operative and broken-broken). The parameters in this example are $\lambda = 1$, $\mu_1 = 5$, $\mu_2 = 2.5$, $\xi_1 = 0.4$, $\xi_2 = 0.2$, $\eta_1 = \eta_2 = 0.1$, $c_1 = c_2 = 1$. The optimal policy was computed by the *Policy Stability* method.

This example shows that the optimal policy does not lend itself to simple characterization. Jobs are not always sent to the shorter queue. Sometimes they are sent to

						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	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1
j_1	10	1	1	1	1	1	1	1	1	1	1	1
	11	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	1	1	1
	13	2	1	1	1	1	1	1	1	1	1	1
	14	2	2	1	1	1	1	1	1	1	1	1
	15	2	2	2	1	1	1	1	1	1	1	1
	16	2	2	2	2	1	1	1	1	1	1	1
	17	2	2	2	2	2	1	1	1	1	1	1
	18	2	2	2	2	2	2	1	1	1	1	1

Table 2.1: Optimal routing decisions: server 1 operative, server 2 broken

a queue where the server is inoperative, even though an operative sever is available. There might be a rule of a 'threshold' type, i.e. 'if the difference between the two queues is greater than a certain value, send the job to the shorter queue'. However, what determines the value of that threshold, and how, is unknown.

In the absence of a characterization, there are clearly great practical difficulties in implementing dispatchers based on table look-up. One would have to pre-compute and store a large number of tables, corresponding to different sets of parameter values, and then decide which table to use, depending on the currently observed conditions. A more practicable approach would be to construct heuristic policies which, while not optimal, perform reasonably well over a wide range of parameter values. Such heuristics are introduced and evaluated in the next chapter.

2.4 Results

The first set of experiments carried out were to compare the three different methods of producing the optimal policy. Due to the comprehensive nature of the optimisation systems, only two servers are being considered. The state space is truncated at $J_1 = J_2 = 100$, giving a total state space of size $400,000$). The run times of the *Cost Iteration*, *Policy Stability* and *Policy Improvement* algorithms, for different values of the discount factor α, are illustrated in Figure 2.2.

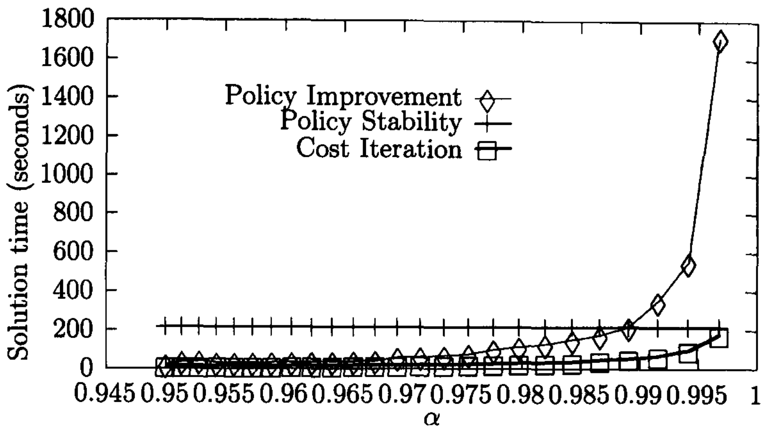

Figure 2.2: Run times of different solution methods

In this model, the two queues differ only in the unit holding costs. The parameters are: $\lambda = 4$, $\mu_i = 5$, $\xi_i = \eta_i = 0.1$ $(i = 1, 2)$, $c_1 = 1$, $c_2 = 5$; i.e., each server is available for half of the time on the average, and the total service capacity is 5. The general pattern of behaviour of the three solution methods' run times do not depend strongly upon the system parameters. For low values of α, *Policy Improvement* and *Cost Iteration* take much less time to solve than *Policy Stability* (which does not depend upon α), but their run times rise to above, at values of α significantly below 1.

These parameters are normalized so that the uniformization constant becomes

$\Lambda = 1$. The termination criterion for *Cost Iteration* is $\epsilon = 0.000001$, while *Policy Stability* terminates when the policy does not change for 100 consecutive iterations. Since that algorithm does not depend on α, it is run only once; the resulting run time is shown as a horizontal line.

Cost Iteration and *Policy Improvement* are very fast for $\alpha < 0.97$, but start slowing down thereafter. The number of iterations performed by *Cost Iteration* varies from 273 to 3637. *Policy Stability* performs 4291 iterations before the policy stabilizes. *Policy Improvement* carries out between 3 and 7 improvement steps, each including the solution of a large set of simultaneous linear equations; that solution is obtained by iterations, with $\epsilon = 0.000001$. However, the coefficient matrix becomes ill-conditioned when $\alpha \to 1$. That explains the steep increase in the run times of *Policy Improvement* when α is very close to 1.

To evaluate the performance of any routing policy, u, we use a single average cost metric, C_u, which is computed as follows. First, find the steady-state distribution, $\pi_u(S)$, of the system state under policy u. This is obtained by solving numerically the balance equations,

$$\pi_u(S) = \sum_{S'} \pi_u(S') q_u(S', S) , \tag{2.13}$$

(where the one-step transition probabilities $q_u(S', S)$ are given by (2.6)), together with the normalizing equation,

$$\sum_{S} \pi_u(S) = 1 . \tag{2.14}$$

The state space is truncated as before.

The average cost incurred under policy u per unit time, C_u, is then given by

$$C_u = \sum_{S} \left[\pi_u(S) \left(\sum_{i=1}^{N} c_i j_i \right) \right] = \sum_{i=1}^{N} c_i L_{u,i} , \tag{2.15}$$

where $L_{u,i}$ is the average number of jobs in queue i under policy u.

Figure 2.3 compares the average costs of the optimal policies returned by the three solution methods, for different values of α. The parameter values are the same as in Figure 2.2.

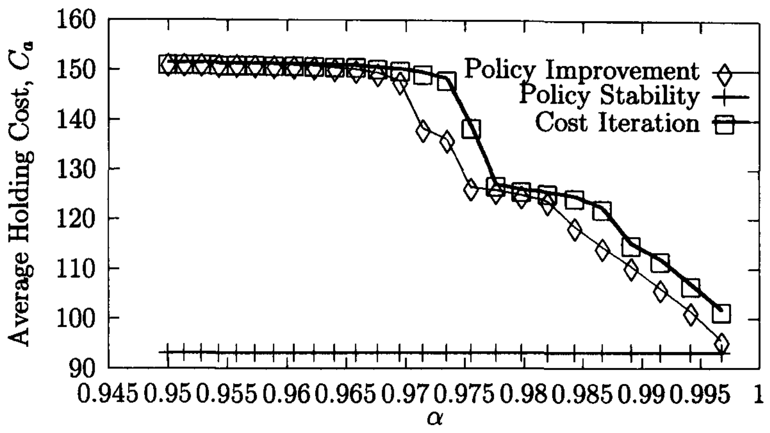

Figure 2.3: Effect of α on optimal policy

The remarkable feature of Figure 2.3 is the strong dependence between α and the performance of the optimal policy. The optimal policy for $\alpha = 1$, returned by the *Policy Stability* algorithm, performs significantly better than the ones for $\alpha < 1$ (the policies returned by *Cost Iteration* and *Policy Improvement* are very similar). The differences in performance between $\alpha = 1$ and $\alpha < 1$ become small only when $\alpha > 0.99$. For these values of α, the necessary solution times for *Policy Improvement* and *Cost Iteration* are much higher than the solution time for *Policy Stability*.

2.5 Conclusions

This section has explored an initial problem in the field of distributed processing and job routing. Three approaches to computing the optimal routing table have been implemented and found to be in agreement both in the general nature of the optimal policy and also in the resulting values of the average holding cost.

It is clear from the description of the algorithms, however, that none of the approaches are suitable for scaling up to systems with tens, or even hundreds of servers. Therefore, there is a need for heuristics which are a good match to the efficiency of allocation of the optimal policy but without the large storage and computational needs of the optimal. This is the subject of the next chapter.

Chapter 3

Servers Subject to Breakdowns and Repairs: Heuristic Policies

3.1 Motivation

The model from the previous chapter is now evaluated using a number of heuristics for the allocation of incoming jobs to servers. Initially these are compared to the optimal policy, evaluated as in the previous chapter, for a small number of servers. However as the computation of the optimal policy becomes overly expensive as the number of servers becomes large, they are later compared with one another in order to find the heuristic which performs best.

3.2 Policies

1. **Random**

   Send incoming jobs to queue i, with probability $1/N$ ($i \in 1, 2, \ldots, N$), regardless of the system state (other than in the cases where queues have reached their truncation lengths).

   This is the heuristic which requires the least state information, as all that is necessary is which servers are able to receive incoming jobs. If the operational parameters are known, they could be, partially or completely, taken into account to create a *Weighted Random* policy which can be expected to perform better, but in this chapter only the unweighted version will be considered.

2. **Selective**

   If no servers are operative, then route to server i with probability $1/N$; otherwise send jobs to server i with probability b_i/O, where

$$O = \sum_i b_i$$

   ($i \in 1, 2, \ldots, N$). This is one of a class of policies discussed in [22].

   This heuristic requires more information than the *Random* heuristic, as it requires, for each server, the current operative state, b_i, but not the queue sizes.

3. **Shortest Queue**

   Send jobs to queue i if $j_i = min_{i'} j_{i'}$. This policy requires the current queue length, j_i for each server, which is the greatest amount of information of all the heuristics so far.

4. **Selfish**

   If a job finds state S on arrival, evaluate the expected non-discounted cost, $d(S, i)$, which it would incur if sent to queue i:

$$d(S,i) = c_i \left[(j_i + 1) \frac{1}{\mu_i} \frac{\xi_i + \eta_i}{\eta_i} + \frac{1 - b_i}{\eta_i} \right].$$

   Where

$$\frac{1}{\mu_i} \frac{\xi_i + \eta_i}{\eta_i}$$

   is the average service period, taking into account the periods of unavailability, and

$$\frac{1 - b_i}{\eta_i}$$

   is the average time remaining of the current, if any, unavailable period.

   Send the job to queue i if $d(S, i) = min_{i'}(d(S, i'))$.

   This heuristic, and the *Index* heuristic following, require full state information for each server.

5. **Index Heuristic**

Whittle's idea [26] was to expand the class of stationary policies by including certain unrealizable policies. Suppose that, at an arrival epoch, a routing function u may add a job to any number of queues simultaneously. Let $\beta_i(u)$ be the resulting steady state arrival rate into queue i. Routing function u is then said to be 'broadly admissible' if the overall arrival rate under u is λ (which, as at some arrival epochs, a job may be added to multiple queues, necessitates that at others the arriving job is added to none, i.e. the job is lost):

$$\sum_{i=1}^{N} \beta_i(u) = \lambda \,,$$

or

$$\sum_{i=1}^{N} [\lambda - \beta_i(u)] = (N-1)\lambda \,. \tag{3.1}$$

Whittle's relaxed optimization problem, which replaces (2.4), is to find

$$\overline{C}^{min} = \inf_u \sum_{i=1}^{N} c_i L_{u,i} \,, \tag{3.2}$$

where the minimum extends over all broadly admissible routing functions. Since that class is wider, we must have $\overline{C}^{min} \le C^{min}$.

The constraint (3.1) can be included in the minimization (3.2) by means of a Lagrangian multiplier, w:

$$\tilde{C}^{min}(w) = \inf_u \sum_{i=1}^{N} (c_i L_{u,i} - w g_u) \,, \tag{3.3}$$

where

$$g_u = \sum_{i=1}^{N} [\lambda - \beta_i(u)] - (N-1)\lambda \,.$$

This last problem can be decomposed into a sum of N separate optimizations:

$$\tilde{C}^{min}(w) = \sum_{i=1}^{N} \tilde{C}_i^{min}(w) - w(N-1)\lambda \, , \tag{3.4}$$

where

$$\tilde{C}_i^{min}(w) = \inf_u [c_i L_{u,i} - w(\lambda - \beta_i(u))] \, . \tag{3.5}$$

Note that the last term in (3.4) does not depend on the policy.

The optimization problem (3.5), referred to as 'problem (i, w)', is set in the context of an isolated single server queue with arrival rate λ, service rate μ_i, and breakdown and repair parameters ξ_i and η_i respectively. The system state is a pair, $S_i = (j_i, b_i)$, where j_i is the number of jobs present and b_i is 0 if server i is inoperative, 1 if operative. The 'routing policy' in this case consists in deciding whether an incoming job which sees state S_i should be accepted or rejected. If accepted, the job incurs a holding cost of c_i per unit time spent in the system; if rejected, it incurs cost w.

It is reasonable to assume that the optimal policy for problem (i, w) is of the threshold type, i.e. there are two integers, $T_0^i(w)$ and $T_1^i(w)$, such that

> An incoming job which finds state $S_i = (j_i, b_i)$ is accepted if either $b_i = 0$ and $j_i \le T_0^i(w)$, or $b_i = 1$ and $j_i \le T_1^i(w)$.

Moreover, since the higher the cost of rejection, the greater the incentive to accept jobs, it is likely that both $T_0^i(w)$ and $T_1^i(w)$ are non-decreasing in w.

It can be shown that, under these monotonicity conditions, there exists a Lagrangian multiplier, w^*, such that the solution to (3.3) is a solution to Whittle's relaxed problem (3.2). Hence there is a solution to Whittle's problem in the form of a superposition of optimal policies for the single queue problems (i, w^*).

Now we can define the Whittle index, $W_i(S_i)$, for queue i in state S_i:

$$W_i(S_i) = \begin{cases} \inf\{w \mid T_0^i(w) \geq j_i\} & \text{if } b_i = 0 \\ \inf\{w \mid T_1^i(w) \geq j_i\} & \text{if } b_i = 1 \end{cases} . \tag{3.6}$$

In other words, the index is the minimum rejection cost that would cause an incoming job to be accepted, when faced with the given queue size and server availability.

Returning to the original routing problem with N queues, the new ***heuristic*** policy works as follows:

Index routing. Send jobs to the queue with the smallest Whittle index, i.e. to queue i if $W_i(S_i) \leq W_k(S_k)$ for all $k = 1, 2, \dots, N$.

For a given set of parameters, the indices are computed numerically, off-line, and tabulated. That computation involves finding, for a given queue and a given w, the optimal pair of thresholds $T_0^i(w)$ and $T_1^i(w)$; the cost of each pair is calculated by solving the finite-state model for that queue. Repeating that procedure for different values of w yields the index corresponding to state S_i, according to (3.6).

Having computed the indices for all states, the index routing policy would be implemented in practice by means of table look-up.

A part of an index table is shown in Table 3.1. It concerns an asymmetric 2-server system. The parameters are: $\lambda = 4$, $\mu_1 = 5$, $\mu_2 = 10$, $\xi_i = 0.3$, $\eta_i = 0.1$, $c_i = 1$ $(i = 1, 2)$. The table shows the queue 1 and queue 2 indices for queue sizes from 0 to 8.

j	0	1	2	3	4	5	6	7	8
W_1	1.00	1.00	5.66	17.29	31.68	50.31	79.91	107.41	144.38
W_2	1.00	1.00	5.66	13.14	20.87	28.05	37.71	39.59	53.22

Table 3.1: Whittle indices for queue 1 and queue 2

There are, of course, many other policies which could have been considered, which may out-perform some or most of the above. The selected set does permit the observation of the variation in performance as more stat information is taken into account.

3.3 Results

3.3.1 Two servers

The results covered in this section will involve a system of two servers, and allows comparison of the optimal policy, calculated using the *Policy Stability* algorithm, with the heuristics described in Section 3.2. For the remainder of this chapter, unless noted otherwise, the holding costs c_i are taken to be 1.

For small to moderate numbers of servers, exact solutions of the balance equations (see Equation 2.13 and Equation 2.14) are computed, giving solutions to the steady state distribution of the system under the given policy, from which the average holding cost can be computed (using Equation 2.15), as in Chapter 2.

Figure 3.1 illustrates the comparison of heuristics with the optimal policy for a model where $\mu_1 = 5$, $\mu_2 = 2.5$, $\eta_1 = \eta_2 = 0.1$, $\xi_1 = 0.4$, $\xi_2 = 0.2$; server 1 is faster but less reliable than server 2. The system is stable when $\lambda < 1.83$.

It can be observed that the dynamic heuristics (the *Shortest Queue*, *Selfish* and *Index*) are close to optimal. The static (*Random*) and semi-static (*Selective*) policies are less efficient, especially as the load increases. In this model, the *Random* policy performs better then the *Selective* policy at heavier loads, which is contrary to expectation, and is probably a feature of the parameters chosen.

The next experiment compares the performance of the optimal and heuristic policies when the asymmetry of the two servers increases. The arrival rate is fixed at

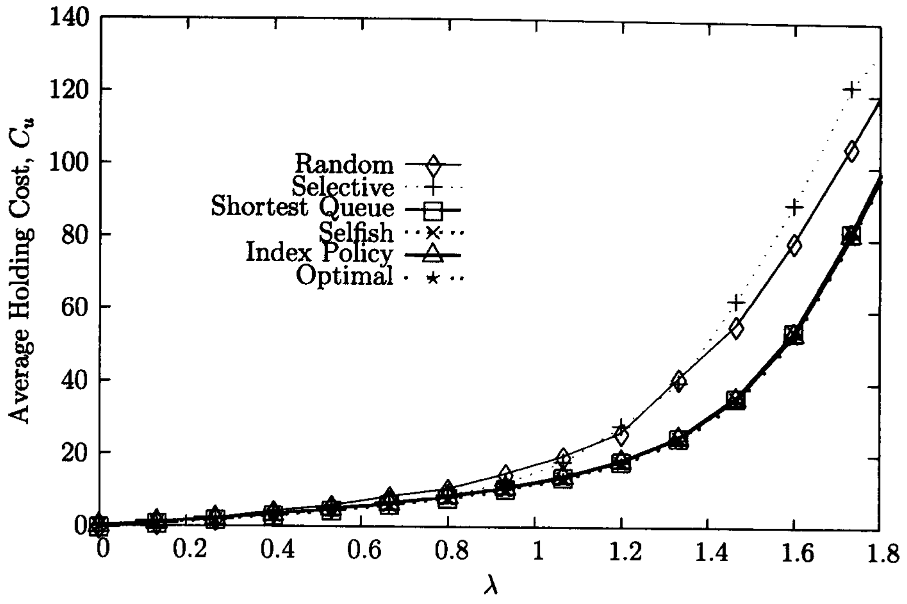

Figure 3.1: Increasing arrival rate

$\lambda = 4$, as is the service rate of server 1, $\mu_1 = 5$, and the breakdown and repair rates, $\xi_1 = \xi_2 = 0.3$, $\eta_1 = \eta_2 = 0.1$. What varies is the service rate of server 2, μ_2. The results are shown in Figure 3.2.

As expected, the average costs decrease as the total service rate increases. The *Selfish* and *Index* heuristics are once again close to optimal (with the *Selfish* slightly more efficient than the *Index* heuristic), but now the *Shortest Queue* policy is significantly poorer, with the *Random* and *Selective* policies much worse.

Another way of increasing the asymmetry between the nodes is to increase the difference between the average lengths of their operative and inoperative periods. This is done in the experiment illustrated in Figure 3.3. The arrival and service parameters are fixed, $\lambda = 4$, $\mu_1 = 5$, $\mu_2 = 25$, as are the breakdown and repair rates of server 1, $\eta_1 = 0.1$, $\xi_1 = 0.3$. The operative and inoperative periods of server 2 increase (ξ and η decrease), while their ratio remains fixed, $\xi_2 = 3\eta_2$.

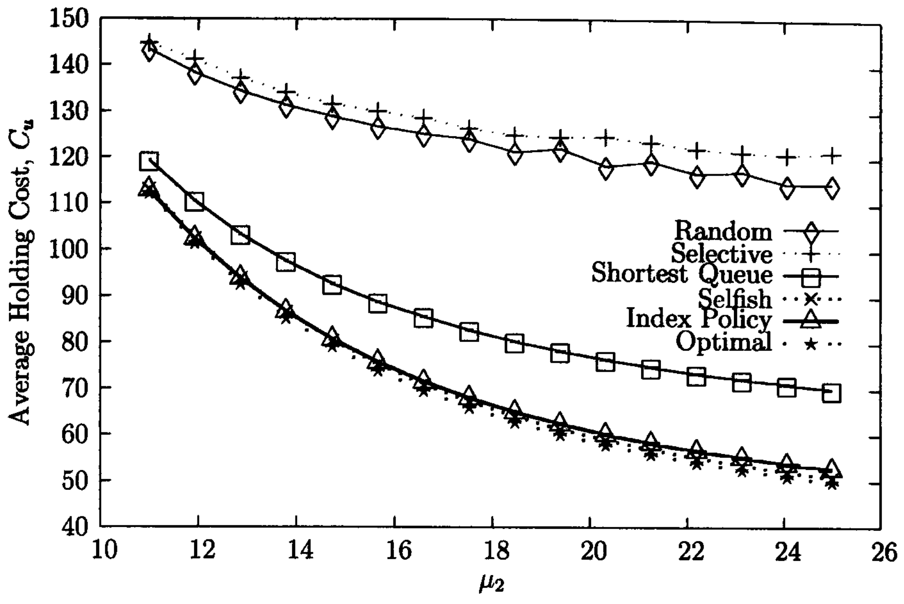

Figure 3.2: Changing service rate at server 2

Note that, although the average service capacity does not change when the operative and inoperative intervals increase in fixed ratio, the average queue sizes, and hence costs, nevertheless increase. This is a known phenomenon. In this case, that increase is slowed as server 1 can absorb some of the load during the long inoperative periods of server 2.

The *Index* policy is a very close match to the *Optimal* policy. At short operative and inoperative periods, the *Selfish* heuristic is near optimal, but as the periods increase, it moves further from optimality. The *Shortest Queue* heuristic, in comparison, is a relatively poor approximation to the *optimal* policy at short periods, but becomes better as the periods are longer. The *Random* and *Selective* heuristics are once again poor matches for the optimal policy over the entire experimental range.

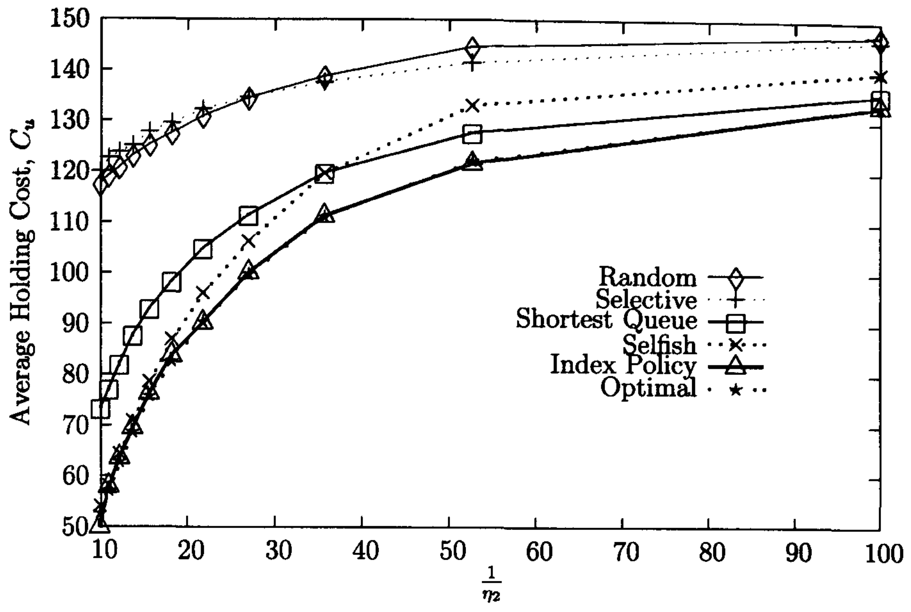

Figure 3.3: Increasing operative and inoperative periods at server 2

3.3.2 More than two servers

In Figure 3.4, the effect of increasing the arrival rate to three servers is shown. All servers are identical, with service rates $\mu_i = 2.2$, breakdown rates $\eta_i = 0.1$ and repair rates, $\xi_i = 0.3$. This gives a maximum stable arrival rate of $\lambda = 1.65$, which is apparent on the figure.

It is clear that the *Random* policy is the furthest from optimal over the range, followed by the *Selective* policy. This is due to the random nature of their allocation procedures, not taking into account the queue lengths of the servers. The *Shortest Queue* policy is a good fit in this experiment, due to the lack of variation between the servers' parameters. However, the best policies are the *Selfish* and *Index* heuristics, with little between them.

In the next experiment, there are two identical servers and the third varies in service rate. The arrival rate is fixed at $\lambda = 4$, as are the service rates for servers

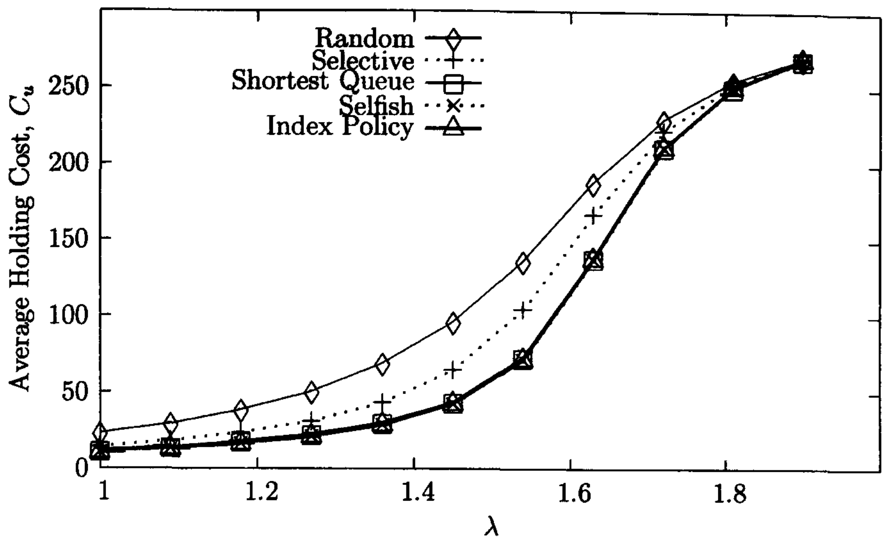

Figure 3.4: Effect of increasing the arrival rate

1 and 2, at $\mu_1 = \mu_2 = 5$, and the breakdown and repair rates of all servers, at $\eta_1 = \eta_2 = \eta_3 = 0.1$ and $\xi_1 = \eta_2 = \eta_3 = 0.3$. The minimum stable value of $\mu_3 = 6$. The results are illustrated in Figure 3.5.

Across the entire experimental range, the *Selfish* heuristic is a very close match to the *Optimal* policy. At low μ_3, the *Shortest Cost* heuristic is a good approximation to the optimal, but as μ_3 increases, it becomes steadily worse until it becomes as poor as the *Selective* policy. For the entire range, the *Random* policy is a poor approximation to the *Optimal*.

Figure 3.6 is essentially the same experiment as carried out in Figure 3.3, but with 3 servers. The difference is that the arrival rate is doubled, $\lambda = 8$, servers 1 and 2 have the same parameters as server 1 did ($\mu_1 = \mu_2 = 5$, $\eta_1 = \eta_2 = 0.1$, $\xi_1 = \xi_2 = 0.3$), and server 3 has twice the service rate as server 2 did, $\mu_3 = 50$. Once again, the operative and inoperative periods are increased in ratio, $\xi_3 = 3\eta_3$.

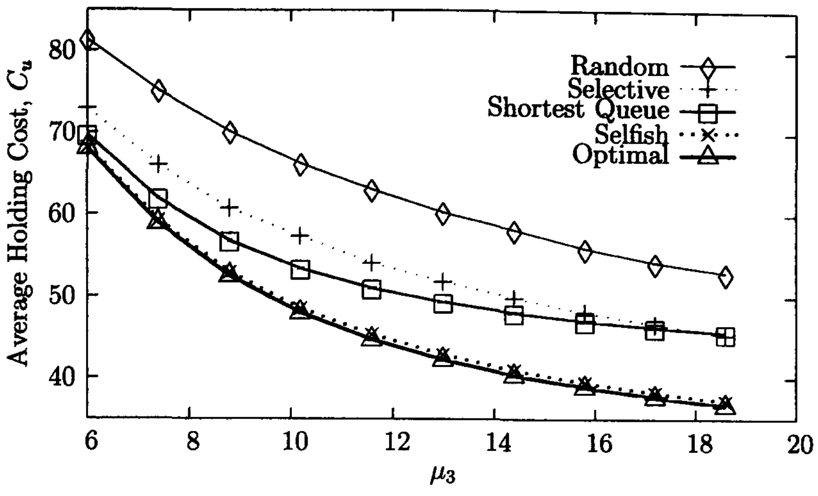

Figure 3.5: Changing service rate at server 3

As expected, the *Selfish* heuristic is close to the *Index* policy at small values of operative and inoperative periods, but becomes a worse fit as the periods increase in length. The other heuristics also behave in the same manner as before.

The remaining experiments consider situations with a much larger number of servers (up to 20). Therefore, the steady state distribution of the system is no longer practical to calculate, and so the distribution as a results of simulation will be used in its place. In addition, the *Optimal* Policy can no longer be computed in a reasonable amount of time and so the heuristic policies will be compared against one another.

Simulation

A simulation of a large number of state transitions can be carried out as follows:

1. Calculate the total instantaneous transition rate, $r(S)$, for the current state, S (see Equation 2.2).

2. Compute the exponentially distributed amount of time the current state has

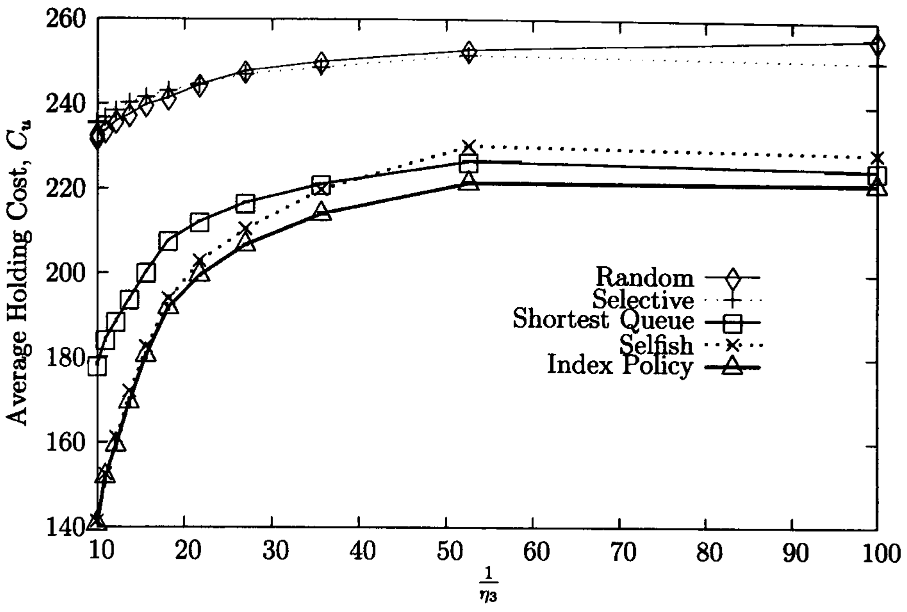

Figure 3.6: Increasing operative and inoperative periods at server 3

been occupied, with a mean of $r(S)$.

3. Increment the logged amount of time the current state has been occupied.

4. Randomly determine the new state, S' according to the system parameters and the allowed transitions from the current state S, with one-step transition probabilities under policy u,

$$q_u(S, S') = r_u(S, S')/r(S)$$

with $r_u(S, S')$ and $r(S)$ given by 2.1 and 2.2 respectively.

Once the simulation is complete, compute $\pi_u(S)$ by dividing each element of the log of occupancy by the total elapsed time. Then, calculate the average cost incurred under policy u per unit time, C_u by

$$C_u = \sum_S \left[\pi_u(S) \left(\sum_{i=1}^{N} c_i j_i \right) \right] , \tag{3.7}$$

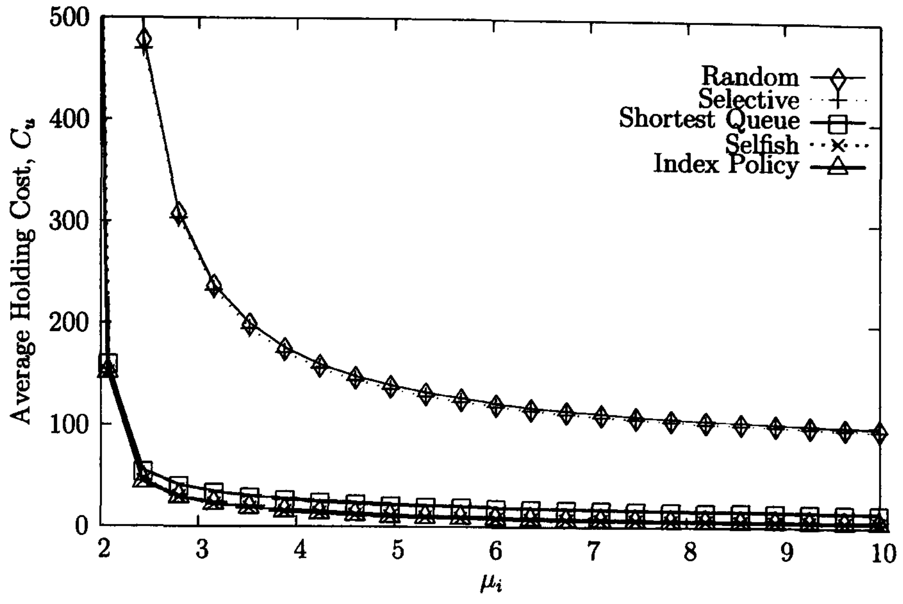

Figure 3.7: Varying service rate of all servers

where j_i is the number of jobs in queue i, in state S.

The first experiment with large N is to vary the service rate. The arrival rate is fixed, $\lambda = 10$, as are the breakdown and repair rates, $\eta = 0.1$, $\xi = 0.3$, of all 20 servers. Figure 3.7 illustrates the result.

The static (*Random*) and semi-static (*Selective*) heuristics remain unstable at μ above the minimum necessary for a balanced system, and as μ increases, are always far worse performed than the dynamic heuristics. Between the dynamic heuristics, the *Shortest Queue* heuristic is the worse performing, as expected, and there is little between the *Selfish* and *Index* heuristics.

An interesting experiment to try is to increase the number of servers in proportion to an increase in arrival rate. The results are illustrated in Figure 3.8, where up to 20 identical servers, with service rate $\mu = 8.8$, and operative and inoperative

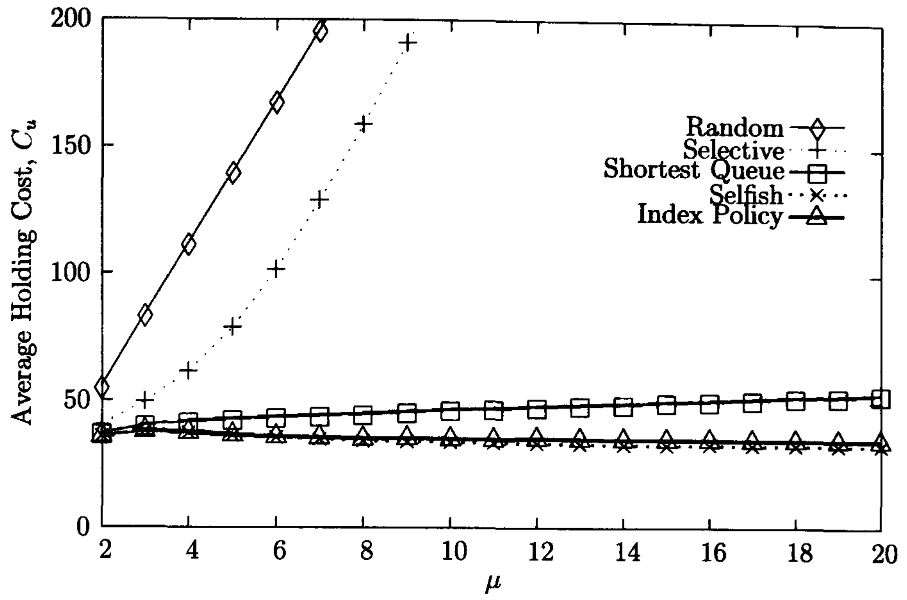

Figure 3.8: Increasing number of servers, with a constant load

rates, $\eta_i = 0.1$, $\xi_i = 0.3$ are served by an incoming stream of jobs with arrival rate $\lambda = 1.4N$, where N is the number of servers. Therefore, the average system load should be constant, regardless of the number of servers, at 64%.

It is clear that the *Random* and *Selective* heuristics are unable to successfully deal with this situation, as the average holding costs rise in approximate proportion to the arrival rate. The remaining heuristics perform much more satisfactorily, with the *Shortest Queue* slowly rising with N, presumably due to the increasing number of jobs routed to inoperative servers. Both the *Selfish* and *Index* heuristics do not significantly change with increasing N, after a small period of adjustment at low N, indicating that they are routing the incoming jobs in an acceptable manner.

In the remaining experiments, there are two groups of nodes, with the same number of servers in each. All the servers in group 1 have the same μ, η and ξ, as do those in group 2.

In Figures 3.9 and 3.10, an interesting variation is considered, where one group of servers becomes faster, but with longer average inoperative periods. Once again, there are 20 servers, with one group having fixed service rate, $\mu_1 = 2.2$, and breakdown and repair rates, $xi_1 = \eta_1 = 0.3$. The other group has a fixed breakdown rate of $\xi_2 = 0.3$, but the service and repair rate are related by $\eta_2\mu_2 = 0.88$.

As each queue is truncated at $j_i = 100$ $(i = 1, 2, \ldots, 20)$, average holding costs close to $2,000$ represent parameter values and heuristics for which most of the queues are saturated for most of the time. From Figure 3.9, it is clear that this condition applies over the entire range for both the Random and Selective heuristics.

The overall system load of this experiment is not a constant, and is plotted in Figure 3.11. This provides one explanation of why the *Random* and *Selective* heuristics perform very badly. As the system as a whole is highly loaded, static and semi-static policies will tend to perform badly.

From Figure 3.10, we can see that the *Shortest Queue* heuristic performs slightly worse than the *Index* and *Selfish* heuristics, with little difference between the latter two.

In Figure 3.12, we see the effect of increasing the service rate of one groups of servers, while keeping the effective service rate (taking into account the operative and inoperative periods) constant. It it clear, once again, that the *Random* and *Selective* policies are greatly suboptimal, with no real difference in their performances.

In Figures 3.13 and 3.14, the service rates for servers in one group are increased, with the breakdown and repair rates also increasing in proportion. There are 20 servers, in two groups. The total arrival rate is fixed at $\lambda = 40$. For the group of servers with fixed parameters, the arrival rate, $\mu_2 = 2.2$, the breakdown rate, $\eta_2 = 0.3$, and the repair rate, $\xi_2 = 0.9$. The system as a whole is stable for $\mu_1 > 14$.

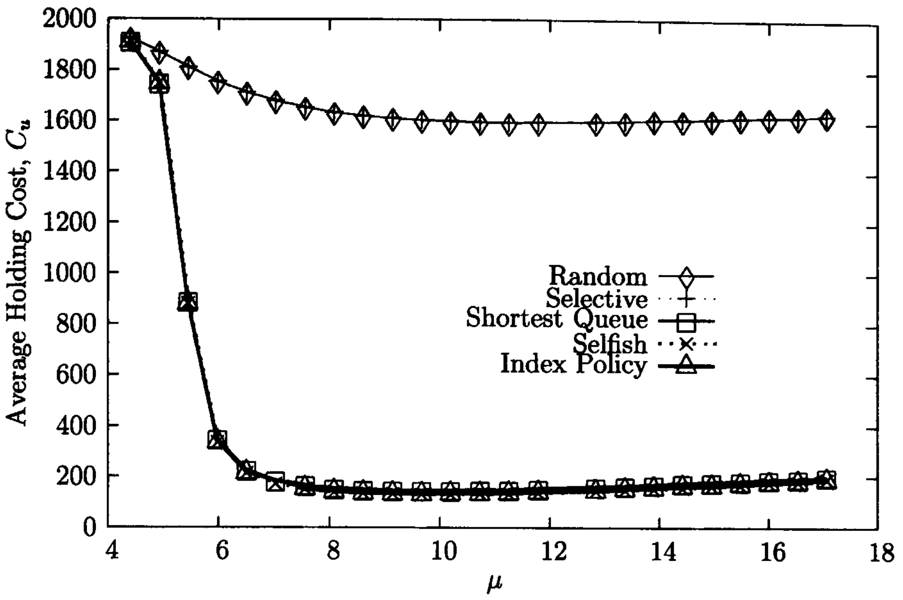

Figure 3.9: The effect of increasing service rate and inoperative periods

From the first graph (Figure 3.13), two observations can be easily made, that there is little difference between the *Random* and *Selective* heuristics, and also that they are both very far from optimal. Considering the detail plot (Figure 3.14), we can see that the *Shortest Queue* policy is quite close to optimal over the considered range, but overall the *Selfish* heuristic is very close to optimal over the entire range.

In Figure 3.15, the arrival rate is varied. There are 20 servers, in two groups. The service rates for both groups are the same, $\mu = 2.2$, and the breakdown and repair rates are very different between the two groups ($\xi_1 = 0.3$, $\xi_2 = 0.03$, $\eta_1 = 0.1$, $\eta_2 = 0.01$), giving rise to the operative and inoperative periods being on average 10 times longer for servers in group 2 as compared to those in group 1. The system is stable for $\lambda < 11$.

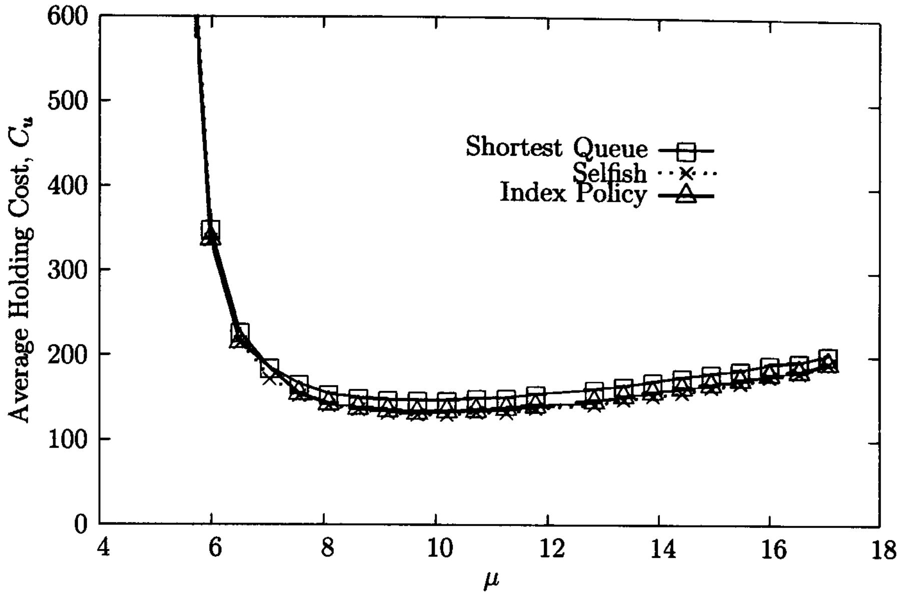

Figure 3.10: The effect of increasing service rate and inoperative periods (detail)

The first observation is that the *Random* and *Selective* heuristics are much worse than the other three, with the *Selective* heuristic slightly outperforming the *Random*, due to the fact that it will not route to inoperative servers when there are operative ones for incoming jobs to go to. The dynamic policies are in close agreement, with the *Shortest Queue* heuristic being slightly the worst, as it will route to the server with the shortest queue, even if it is inoperative, the effect of which will be exacerbated by the possibility that that server is in group 2, and hence likely to remain inoperative for a significant amount of time.

3.4 Conclusions

In this chapter, a number of heuristic policies for routing jobs between servers have been evaluated, and two policies have been found which consistently perform close to the optimal policy in experiments where there are a small number of servers.

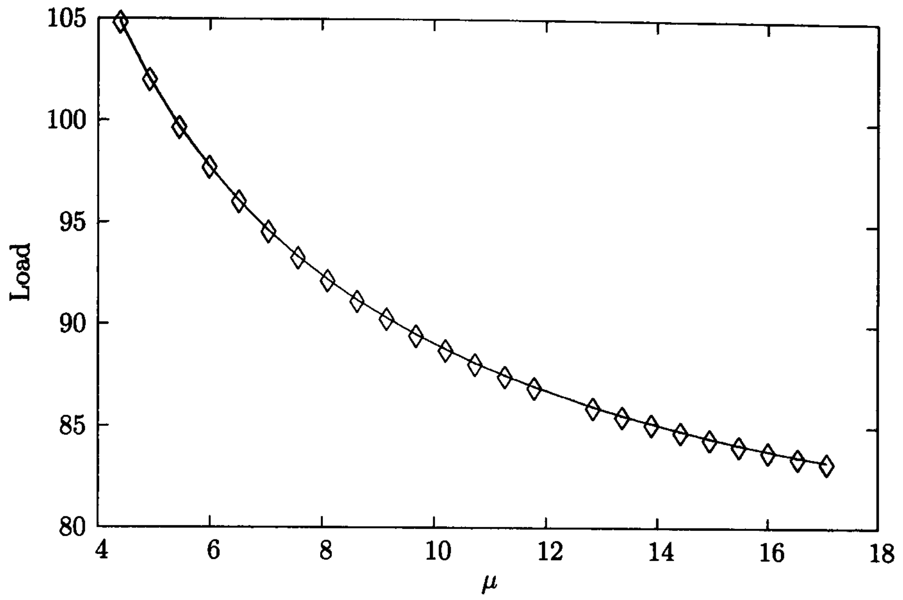

Figure 3.11: The effect of increasing service rate and inoperative periods (System load)

In addition, these policies always perform as well as all other policies under consideration, and usually perform much better. The *Index* Policy requires an initial set-up phase, and storage of the form NM, where N is the number of servers, and M is the desired queue truncation size. Therefore, it is suitable for systems where the configuration does not vary very often, and which has sufficient storage space to store the indices. The benefit is the ease of determining which server to route an incoming job to. The *Selfish* Policy requires no set-up time, and no additional storage, other than the system state and parameters, but must compute the expected non-discounted cost for each server, adding a significant overhead.

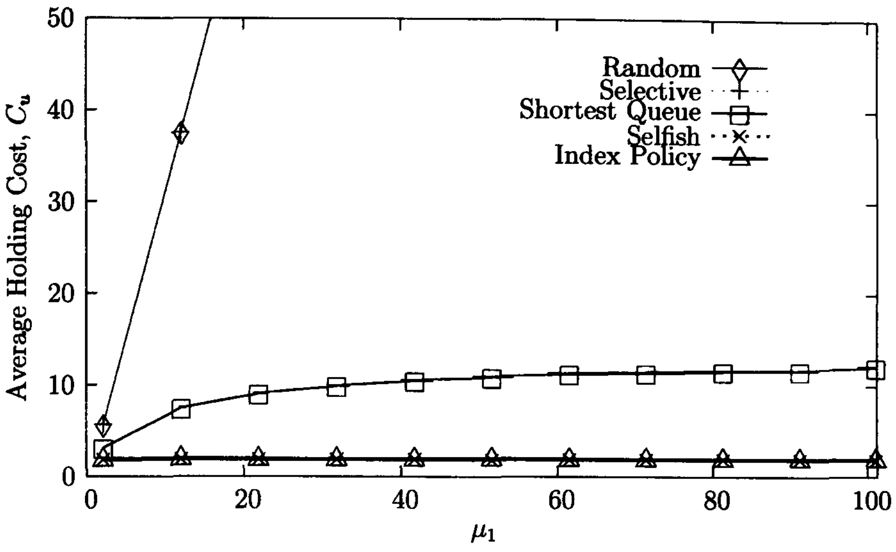

Figure 3.12: The effect of increasing service rate, while keeping effective service rate constant

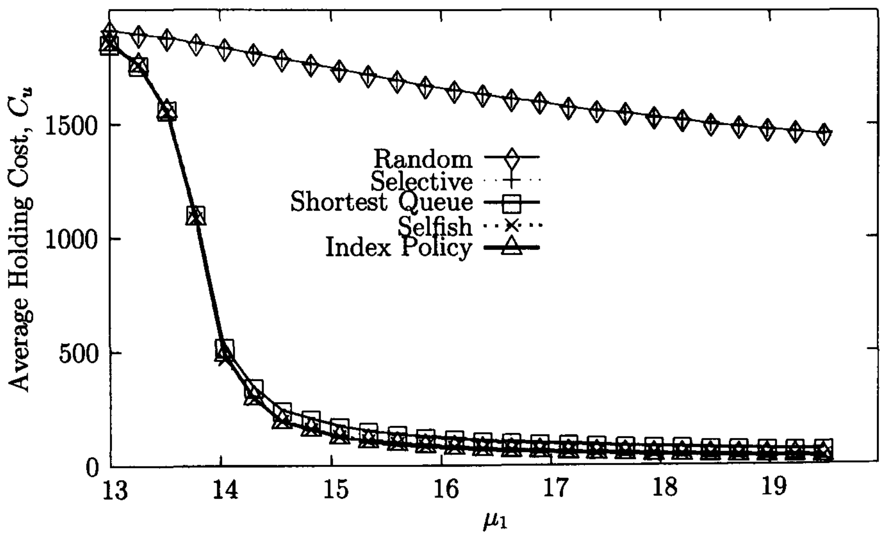

Figure 3.13: Effect of increasing service rate and breakdown rates, in proportion

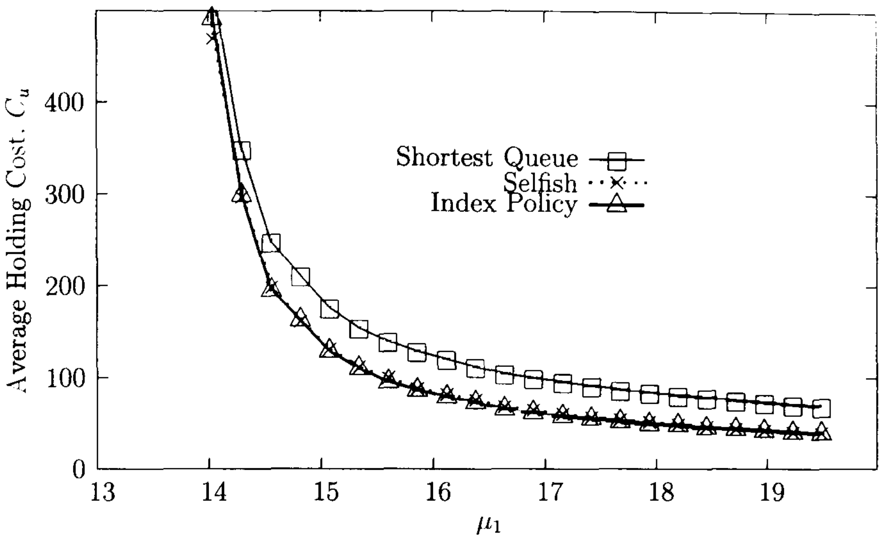

Figure 3.14: Effect of increasing service rate and breakdown rates, in proportion (detail)

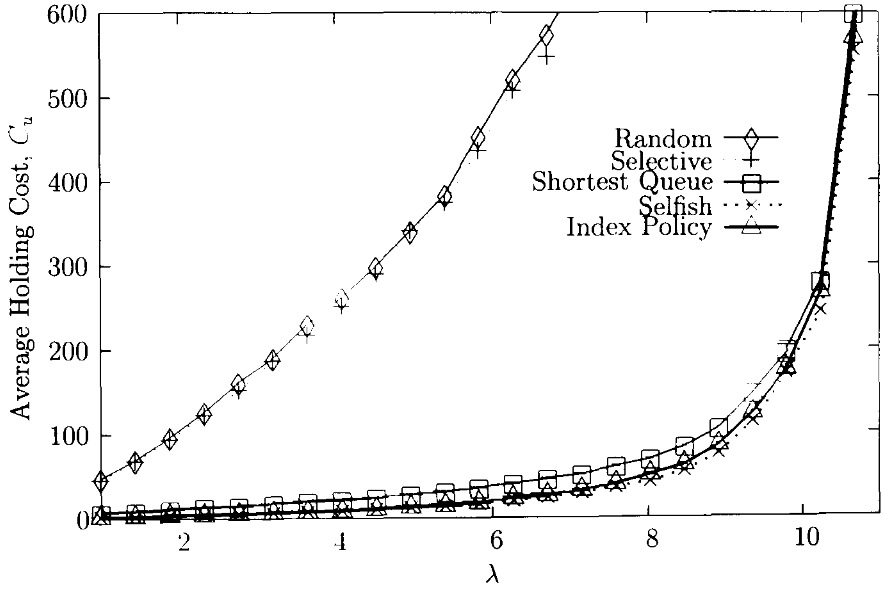

Figure 3.15: Increasing Arrival Rate

Chapter 4

Two Servers with One-Way Transfers

4.1 Motivation

The context of this chapter is once again the need to balance the offered load between servers in the system. In this case the jobs are directly submitted to one of the servers in the system, where it then either waits to commence service, or transfers to another from where it expects to complete service sooner.

Posed in its full generality, this is a complex problem, which is unlikely to yield an exact and explicit solution. This chapter considers a simpler problem of the same form, where transfers are only permitted from server 1 to server 2. This system can then be solved exactly. An approximate solution approach is also developed, which is evaluated and found to be a good approximation to the exact solution.

4.2 Model

Jobs arrive into single-server, unbounded queues, 1 and 2, in independent Poisson streams with rates λ_1, and λ_2, respectively. The required service times at the two servers are independent, exponentially distributed random variables with means $1/\mu_1$, and $1/\mu_2$, respectively. Each job entering queue 1 is assigned, on arrival, an independent timeout period, which is distributed exponentially, with mean $1/\psi$. Any job whose timeout period expires before it commences services is instantaneously transferred to server 2. Therefore, if there are i jobs in queue 1 ($i > 1$), the rate of transfers to queue 2 is $(i-1)\psi$. The system is illustrated in Figure 4.1.

Define L_1 and L_2 to be the steady state numbers of jobs in queue 1 and 2, respectively.

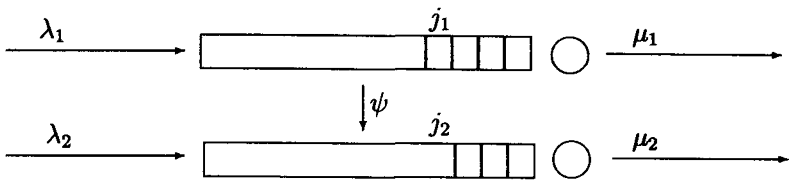

Figure 4.1: Two queues with deadline-driven transfers

tively. Then, we split L_2 into L_{21} and L_{22}, where L_{21} is the average number of jobs in queue 2 that were transferred from queue 1, and L_{22} is the average number of jobs in queue 2 that originally arrived at queue 2. These three properties are the system performance measures. The optimization problem consists of finding the value of ψ that minimizes a cost function of the form

$$C = \min_{\psi} \left(c_1 L_1 + c_2 L_{21} + c_3 L_{22}\right) \ , \tag{4.1}$$

where c_1, c_2 and c_3 are given holding costs. For example, if $c_1 = c_2 = c_3 = 1$, then the objective is to minimize the total average number of jobs in the system. If $c_1 = c_2 = 1$ and $c_3 = 0$, then we want to minimize the average number of jobs originally submitted to queue 1.

To determine the performance measures, it is necessary to find the joint distribution of the two queue sizes. Let $p_{i,j}$ be the steady-state probability that there are i jobs in queue 1, and j jobs in queue 2. These probabilities satisfy the balance equations:

$$\begin{aligned}&[\lambda_1 + \lambda_2 + \mu_1\delta(i>0) + \mu_2\delta(j>0) + (i-1)\psi\delta(i>0)]\,p_{i,j} \\ &= \lambda_1 p_{i-1,j} + \lambda_2 p_{i,j-1} + \mu_1 p_{i+1,j} + \mu_2 p_{i,j+1} + i\psi p_{i+1,j-1}\ ,\end{aligned} \tag{4.2}$$

where all probabilities with a negative index are 0 by definition. The boolean indicator function, $\delta(B)$ is 1 if B is true, 0 otherwise.

Introducing the generating function

$$g(x,y) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} p_{i,j}x^i y^j\ , \tag{4.3}$$

the balance equations can be re-written as

$$\begin{aligned}&\left[\lambda_1(1-x) + \lambda_2(1-y) + \mu_1(1-\frac{1}{x}) + \mu_2\left(1-\frac{1}{y}\right) - \psi\left(1-\frac{y}{x}\right)\right] g(x,y) \\ &= \left[\mu_1\left(1-\frac{1}{x}\right) - \psi\left(1-\frac{y}{x}\right)\right] g(0,y) + \mu_2\left(1-\frac{1}{y}\right) g(x,0) + \psi(y-x)\frac{\partial g(x,y)}{\partial x}\ .\end{aligned} \tag{4.4}$$

This equation, together with the fact that $g(x,y)$ is an analytic function in the interior of the unit disc which satisfies the normalizing condition,

$$g(1,1) = 1\ , \tag{4.5}$$

should in principle determine $g(x,y)$ and hence the joint distribution of the two queue sizes. Unfortunately we have not been able to find the solution.
However, useful information can be generated from 4.4. Setting $y = 1$ gives an equation for the marginal distribution of queue 1:

$$(\lambda_1 x + \psi - \mu_1)\, g(x,1) + (\mu_1 - \psi)\, p_{0,\cdot} = \psi x \frac{\partial g(x,1)}{\partial x}\ , \tag{4.6}$$

where $p_{0,\cdot} = g(0,1)$ is the marginal probability that queue 1 is empty. This is an ordinary first-order linear differential equation, which can be solved in closed form.

Defining $u(x) = g(x,1)$, we can re-write equation 4.6 as

$$u'(x) + \left[\frac{\mu_1 - \psi}{\psi x} - \frac{\lambda_1}{\psi}\right] u(x) + \frac{\psi - \mu_1}{\psi x} p_{0,\cdot} = 0 \tag{4.7}$$

However, it should be emphasized that a non-negative solution exists for all values of the parameters, as long as $\psi > 0$. Hence, queue 1 is stable whenever $\psi > 0$ (this is also intuitively obvious, since the job transfer mechanism prevents queue 1 from growing too large). The probability $p_{0,\cdot}$ is determined from the condition that $g(x,1)$ is finite at $x = 0$; this gives

$$p_{0,\cdot} = \left[1 + \frac{\lambda_1}{\mu_1} + \frac{\lambda_1^2}{\mu_1 \psi} \int_0^1 x^{\frac{\mu_1}{\psi}} e^{-\frac{\lambda_1}{\psi}(x-1)} dx\right]^{-1} . \tag{4.8}$$

Alternatively, the marginal probabilities $p_{i,\cdot}$ can be obtained numerically from the Birth-and-Death equation for queue 1 (see [2]):

$$\lambda_1 p_{i,\cdot} = (\mu_1 + i\psi) p_{i+1,\cdot} \ ; \ i = 0, 1, \dots , \tag{4.9}$$

together with the normalizing equation.

The performance measure L_1 can be expressed in terms of $p_{0,\cdot}$. Setting $x = 1$ in (4.6) gives

$$L_1 \psi = \lambda_1 - (\mu_1 - \psi)(1 - p_{0,\cdot}) . \tag{4.10}$$

This yields a finite value for L_1, provided that $\psi > 0$.

Next, we find the stability condition for queue 2. Consider the distribution of the total number of jobs in the system, whose generating function is obtained by setting $x = y$ in (4.4). After simplification, this gives

$$[\mu_1 + \mu_2 - (\lambda_1 + \lambda_2)x] g(x,x) = \mu_1 g(0,x) + \mu_2 g(x,0) . \tag{4.11}$$

Now set $x = 1$ in (4.11):

$$\mu_1 + \mu_2 - \lambda_1 - \lambda_2 = \mu_1 p_{0,\cdot} + \mu_2 p_{\cdot,0} \,. \tag{4.12}$$

Since queue 2 is stable if, and only if, the probability that it is empty is non-zero, i.e. $p_{\cdot,0} > 0$, we have the following ergodicity condition:

$$\lambda_1 + \lambda_2 < \mu_1(1 - p_{0,\cdot}) + \mu_2 \,. \tag{4.13}$$

This condition depends on the value of ψ through $p_{0,\cdot}$. When (4.13) is satisfied, the performance measures L_{21} and L_{22} are finite.

It is worth pointing out, for future reference, that the difference, $\lambda_{1,2}$, between the arrival rate into queue 1 and the service completion rate at server 1,

$$\lambda_{1,2} = \lambda_1 - \mu_1(1 - p_{0,\cdot}) \,, \tag{4.14}$$

represents the average number of jobs transferred from queue 1 to queue 2 per unit time. That 'internal' traffic rate is strictly positive when $\psi > 0$.

It is possible to express L_{21} and L_{22} in terms of the total average number of jobs in queue 2, L_2. Note that, according to the PASTA property of the Poisson process, a job arriving *externally* into queue 2 sees, on the average, L_2 jobs there, excluding itself (this is not true for the jobs transferring from queue 1). Hence, the average response time of an *external* arrival into queue 2 is $(L_2 + 1)/\mu_2$. By Little's theorem, the average number of jobs in queue 2 that arrived into it from outside is equal to

$$L_{22} = (L_2 + 1)\frac{\lambda_2}{\mu_2} \,. \tag{4.15}$$

The average number of jobs in queue 2 that arrived into it from queue 1 is therefore given by

$$L_{21} = L_2 - L_{22} = L_2(1 - \frac{\lambda_2}{\mu_2}) - \frac{\lambda_2}{\mu_2} \,. \tag{4.16}$$

Thus, the problem reduces to being able to compute L_2.

4.3 Exact and approximate solutions

Since queue 1 is always stable, it can always be truncated without a significant loss of accuracy. Choose an arbitrary error bound, $\epsilon > 0$. Using (4.8) and (4.9), find an integer N such that

$$\sum_{i=0}^{N} p_{i,\cdot} > 1 - \epsilon \, . \tag{4.17}$$

That is always possible. Then, if queue 1 is truncated at threshold N, i.e. new arrivals are not allowed to join it when there are N jobs present, only states with a total probability less than ϵ will become unreachable.

With that truncation, queue 2 can be treated as an unbounded queue whose instantaneous transition rates are modulated by a finite state Markovian environment (the size of queue 1). It can then be solved by spectral expansion [14, 15]. Define the row vectors of probabilities corresponding to states with j jobs in queue 2:

$$\mathbf{v}_j = (p_{0,j}, p_{1,j}, \ldots, p_{N,j}) \; ; \; j = 0, 1, \ldots \, . \tag{4.18}$$

Defining A as the transition matrix of purely lateral transitions

$$\begin{bmatrix} 0 & \lambda_1 & 0 & \cdots & 0 \\ \mu_1 & 0 & \lambda_1 & \cdots & 0 \\ 0 & \mu_1 & 0 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \lambda_1 \\ 0 & 0 & \cdots & \mu_1 & 0 \end{bmatrix} , \tag{4.19}$$

B as the transition matrix of one-step upwards transitions

$$\begin{bmatrix} \lambda_2 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & \psi & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & (N-1)\psi & \lambda_2 \end{bmatrix} , \tag{4.20}$$

and C as the transition matrix of one-step downwards transitions

$$\begin{bmatrix} \mu_2 & 0 & 0 & \cdots & 0 \\ 0 & \mu_2 & 0 & \cdots & 0 \\ 0 & 0 & \mu_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & \mu_2 \end{bmatrix} . \tag{4.21}$$

Let D^A, D^B and D^C be diagonal matrices, defined by their diagonal elements as,

$$D^A(i,i) = \sum_{k=0}^{N} A(i,k); \; D^B(i,i) = \sum_{k=0}^{N} B(i,k); \; D^C(i,i) = \sum_{k=0}^{N} C(i,k) . \tag{4.22}$$

Then, the solution of the balance equations (4.2) has the form

$$\mathbf{v}_j = \sum_{k=1}^{N+1} \alpha_k \mathbf{u}_k x_k^j \; ; \; j = 0, 1, \dots \, , \tag{4.23}$$

where x_k are the eigenvalues of a matrix polynomial, Q, in the interior of the unit disc, $\mathbf{u}_k$ are the corresponding left eigenvectors, and α_k are some (possibly complex) constants, where

$$Q(x) = B + (A - D^A - D^B - D^C)x + Cx^2 . \tag{4.24}$$

The unknown coefficients α_k are determined from the balance equations for $j = 0$ (which have not been used in the matrix polynomial) and the normalizing equation.

Having evaluated the spectral expansion solution, the marginal distribution of the number of jobs in queue 2 is given by

$$p_{\cdot j} = \sum_{k=1}^{N+1} \alpha_k (\mathbf{u}_k \mathbf{e}) x_k^j \; ; \; j = 0, 1, \dots \, , \tag{4.25}$$

where $\mathbf{e}$ is a column vector with $(N+1)$ elements, all of them equal to 1. The average number of jobs in queue 2 is equal to

$$L_2 = \sum_{k=1}^{N+1} \alpha_k(\mathbf{u}_k\mathbf{e}) \frac{x_k}{(1-x_k)^2} . \tag{4.26}$$

All performance measures are now available and can be computed for different values of the control variable, ψ, in order to find the optimum. However, although the numerical solution is not difficult to implement, is fast, and yield accurate results in most cases of interest, there are parameter values which cause numerical problems. In particular, when the truncation threshold N is large, and ψ is large, the matrix whose eigenvalues need to be computed tends to become ill-conditioned.

To cope with these difficulties, and to provide a 'rough-and-ready' alternative to the full solution, we propose a simple approximation which requires almost no computational effort.

Poisson approximation

Assume that the instants of job transfers from queue 1 to queue 2 form a Poisson process, with rate given by (4.14). Then queue 2 can be treated as an isolated M/M/1 queue with arrival rate $\lambda_2 + \lambda_{1,2}$ and service rate μ_2. That queue is stable when (4.13) is satisfied, and its average queue size is equal to

$$L_2 = \frac{\lambda_2 + \lambda_{1,2}}{\mu_2 - \lambda_2 - \lambda_{1,2}} . \tag{4.27}$$

Intuitively, it can be expected that the Poisson approximation will underestimate the average size of queue 2. This is because the assumption of a constant rate of transfer eliminates some of the variability of the transfer process. The smaller the value of ψ, the less bursty the transfers and therefore the more accurate the approximation will be.

4.4 Results

The first experiment consists of evaluating the cost function, $C = L_1 + L_{21} + L_{22}$ (total average number of jobs in the system), for different values of ψ, while keeping the arrival and service rates fixed. The offered loads at queue 1 and queue 2 are 0.9 and 0.5, respectively ($\lambda_1 = 2.7, \lambda_2 = 1, \mu_1 = 3, \mu_2 = 2$). Therefore, we can expect that performance would benefit from some transfers.

In Figure 4.2, the exact values of the cost function are compared to those obtained from the Poisson approximation. It is clear that the Poisson approximation is quite accurate. At low values of ψ, it is almost exact, but the expected underestimation becomes more apparent as ψ increases. The optimal value of ψ is approximately 0.2, according to the exact solution, which is small compared to the service rate at server 1. The optimum value of ψ obtained from the Poisson approximation is 0.25, which is not too dissimilar to the exact value.

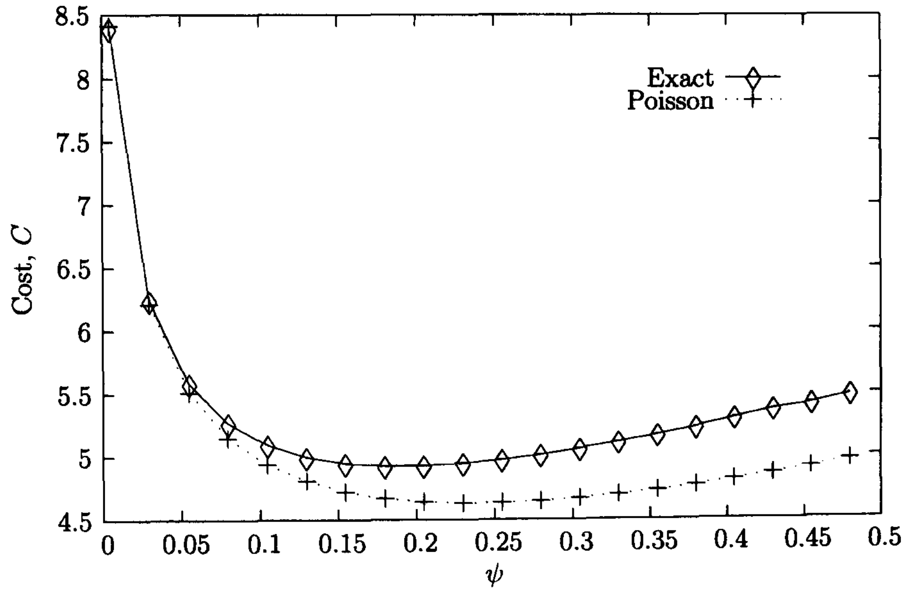

Figure 4.2: Cost as a function of ψ, $C = L_1 + L_{21} + L_{22}$

The error comes from the assumption in the Poisson approximation that the transfer rate is constant relative to the length of queue 1, which is not true. Therefore, as noted above the Poisson approximation will tend to underestimate the length of queue 2, and hence will predict a larger than ideal 'optimal' value of ψ.

In Figure 4.3, the same experiment is carried out once more, except that this time, the cost function is the average number of jobs originally arriving into queue 1, $C = L_1 + L_{21}$. In this case, as we are not considering the average number of jobs which originally arrived into queue 2, we would expect the optimal value of ψ to be greater than before. For this experiment, the exact solution produces an optimal value of ψ of approximately 0.35. As before, the Poisson approximation predicts an optimal ψ that is slightly larger, at 0.4.

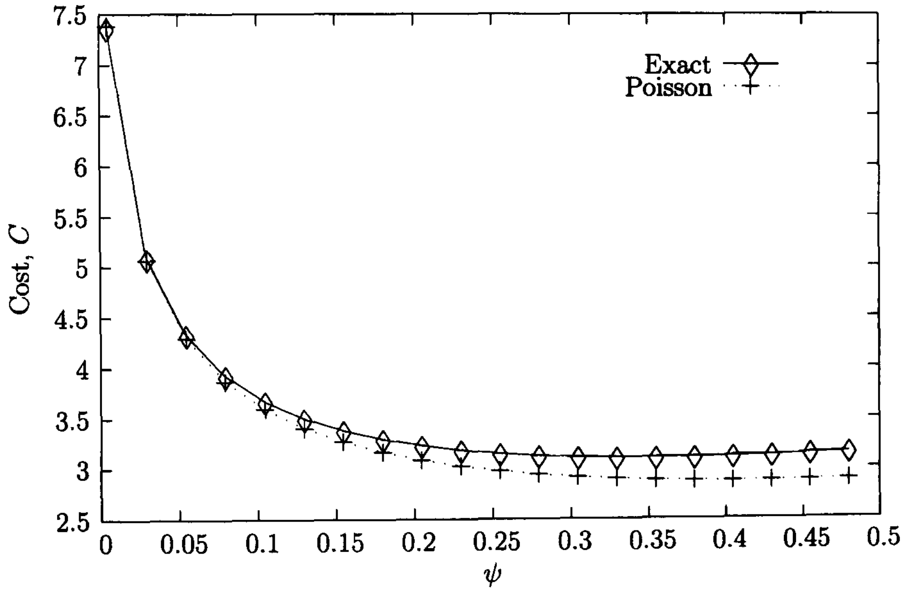

Figure 4.3: Cost as a function of ψ, $C = L_1 + L_{21}$

Another noticeable difference to the previous figure, is the much shallower rise in C with ψ above optimal. This is not unexpected, as the effect of transferring too many jobs from queue 1 to queue 2 will mostly be to increase the amount of time incoming jobs to queue 2 will have to wait.

Figures 4.4 and 4.5 illustrate an experiment where λ_1 varies, where the cost function is the total average number of jobs in the system. The static parameters are the arrival rate into queue 2, $\lambda_2 = 1$, and the service rates of queues 1, $\mu_1 = 3$, and 2, $\mu_2 = 2$.

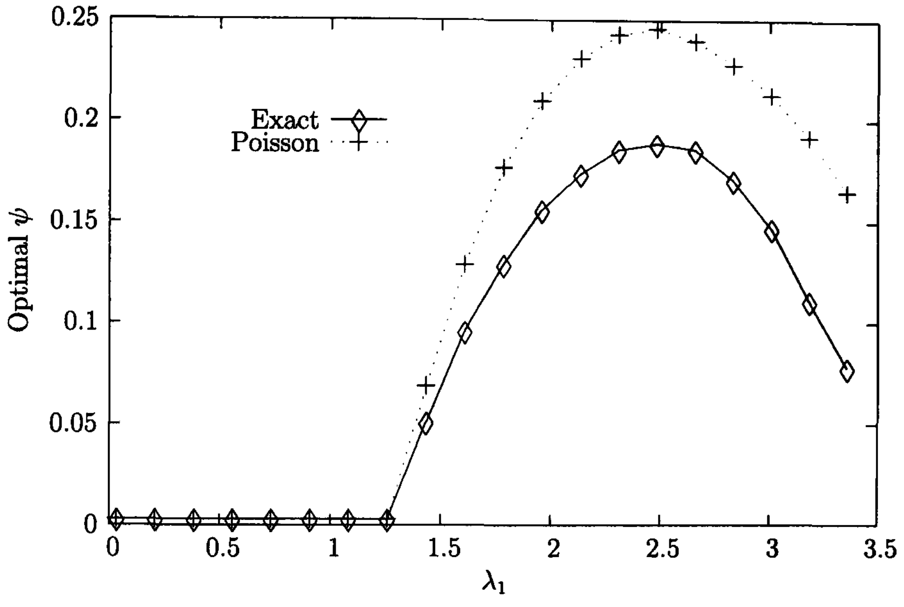

Figure 4.4: Optimal ψ as a function of λ_1, $C = L_1 + L_{21} + L_{22}$

Figure 4.4 shows the value of ψ which each of the solvers calculate to be the optimal value. We can easily see that as before, the optimal values of ψ computed by each solver differ, in this case by a maximum of approximately 0.7. At the point $\lambda_1 = 2.7$, we can see the the difference in values of ψ is around 0.7, which is the same value as the separation between minima observed in Figure 4.2. Both solvers exhibit the same behaviour, with zero transfers for a low arrival rate, rising to a maximum at $\lambda \approx 2.5$ and then decreasing, as the system as a whole becomes highly loaded.

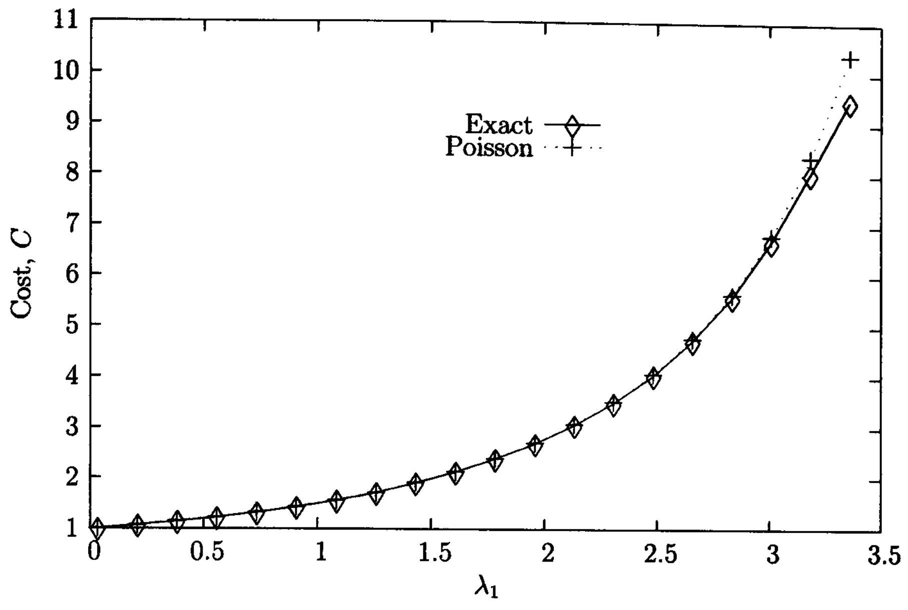

Figure 4.5: Cost as a function of λ_1, $C = L_1 + L_{21} + L_{22}$

In Figure 4.5, the same experiment is carried out, but now the cost function, $C = L_1 + L_{21} + L_{22}$, is evaluated for each of the solvers. From this, it can be seen that in spite of the differing values of ψ, the exact cost of the Poisson approximation is indistinguishable from the cost of using the value of ψ computed with the exact solver, until the system as a whole becomes close to saturation.

When the system becomes close to saturation, the effect of the discrepancy between the optimal value of ψ and that calculated by the approximation becomes greater, and the percentage error in achieved costs increases towards saturation. The same behaviour is observed in the case where the cost function is the average number of jobs originally submitted to queue 1, $C = L_1 = L_{21}$.

In Figures 4.6 and 4.7, we can see the benefit that an optimal value of ψ has upon the cost function. The cost values achievable by the optimal policy are compared with transferring too few jobs, $\psi = 0$, or too many, $\psi = \mu_1$. Note that setting $\psi = \mu_1$ makes queue 1 behave as an M/M/∞ queue.

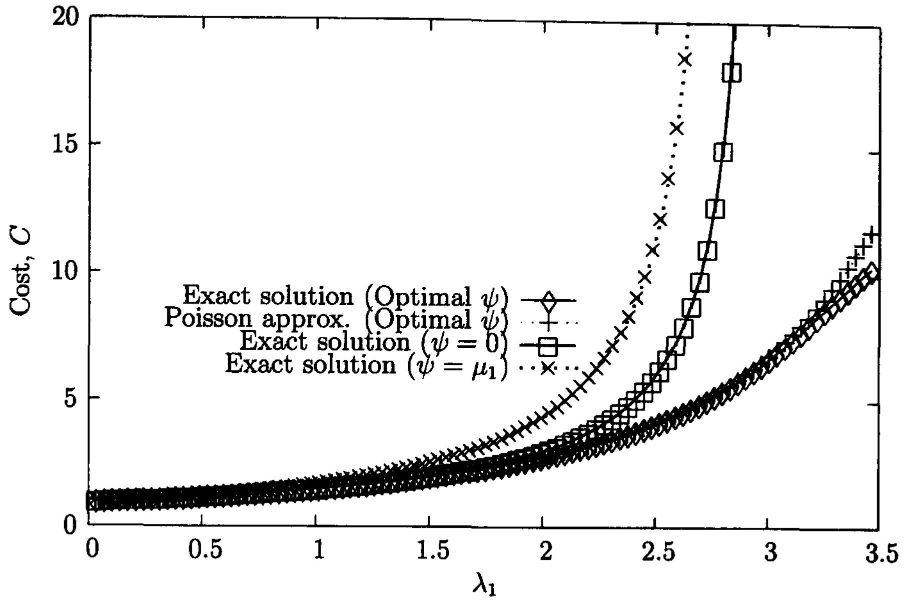

Figure 4.6: The benefits of optimization, $C = L_1 + L_{21} + L_{22}$

Once again, the arrival rate into server 1, λ_1 is varied, and the other system parameters are held constant.

As expected, when the offered load is low, there is little benefit to optimizing the transfer rate. However, as the load increases, a well-chosen ψ can make a big difference. Under the policy $\psi = 0$, queue 1 saturates at $\lambda_1 = 3$; when $\psi = \mu_1$, queue 2 saturates at approximately $\lambda_1 = 2.9$. Alternatively, using a well-chosen ψ (either the optimal value, or the value obtained using the Poisson approximation) keeps both queues stable for offered loads of up to approximately $\lambda_1 = 3.99$. This cannot be significantly improved upon, as $\lambda_2 = 1$, and the total available service capacity, $\mu_1 + \mu_2 = 5$. Note once again that the costs of the policy suggested by the Poisson approximation are almost optimal over the entire range of λ_1, despite the

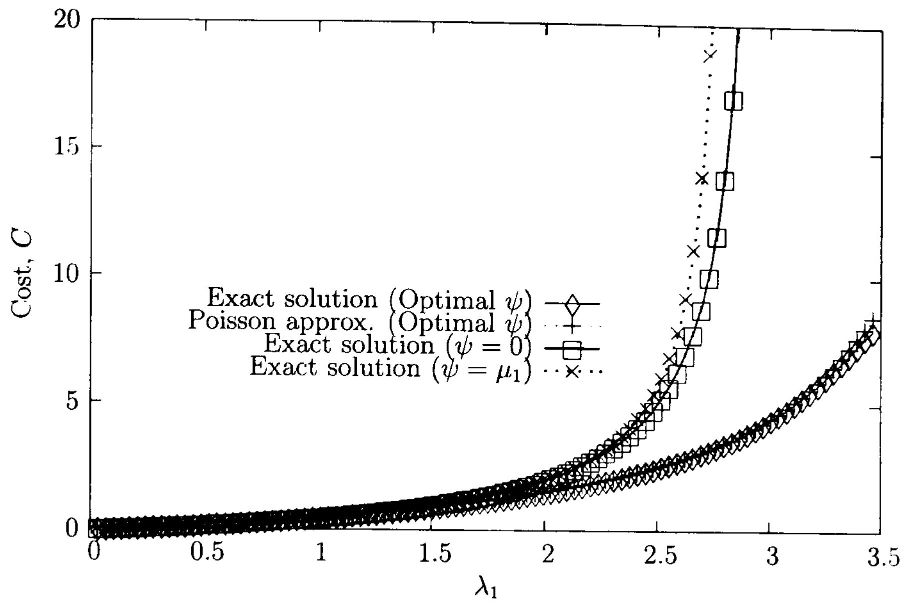

Figure 4.7: The benefits of optimization, $C = L_1 + L_{21}$

overestimation of the optimal ψ.

4.5 Conclusions

This chapter has shown that a simple approximation is a good fit for the simple two-server system with transfers, when compared with the exact solution obtained by solving the balance equations using the spectral expansion technique.

The approximation is a very good fit across most system parameters. The exception is when the system is highly loaded. In this case, the discrepancy between the optimal transfer rate and that obtained from the approximation causes the optimal policy to noticeably out-perform the Poisson approximation.

It is apparent that, for most system parameters, the simple and computationally less intensive approximation will perform adequately for computing a good value for a suggested transfer rate in this simple two server case.

This work will be expanded on in the next chapter where systems of servers with transfers will be considered which are not explicitly solvable.

Chapter 5

Servers with General Job Transfers

5.1 Motivation

This chapter will build upon the model considered in the previous chapter, but instead of transfers only allowed in one direction between two servers, they are now permitted between any servers in a system of many more than two servers.

This leads to the conclusion that the system will not be exactly solvable and so an iterative approximation technique will be used to find the distribution of jobs in the system as a whole, and hence evaluate a cost function.

5.2 Model Definition

Suppose now that the system consists of N servers and queues, numbered $1, 2, \ldots, N$. Jobs arrive *externally* into queue i in an independent Poisson stream with rate λ_i. The required service times at server i are distributed exponentially with mean $1/\mu_i$. Each job joining queue i, whether externally or from another queue, is assigned an independent time-out period which is distributed exponentially with mean $1/\varphi_i$. Any job whose time-out period expires before it reaches the server is instantaneously transferred from queue i to queue j with probability $q_{i,j}$. The system is illustrated in figure 5.1.

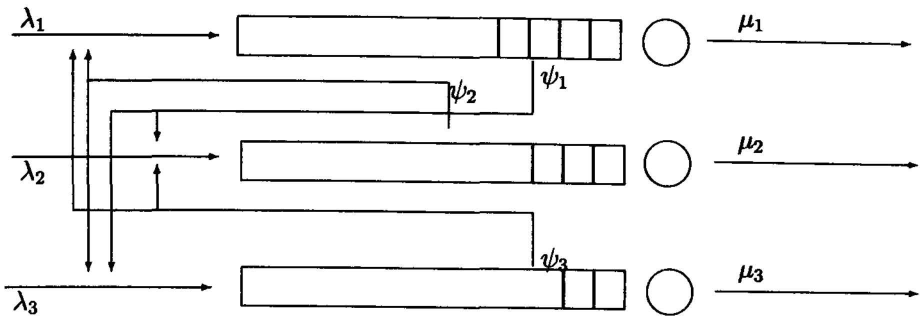

Figure 5.1: 3 servers with job transfers

A matrix of transfer probabilities, $Q = \{q_{i,j}\}_{i,j=1}^{N}$, all of whose row sums are equal to 1, defines a 'transfer policy' for the system. We consider policies which transfer from queue i either all jobs that time out in it, or none of them. In other words, either $q_{i,i} = 0$ (and the other probabilities in row i add up to 1), or $q_{i,i} = 1$ (and $q_{i,j} = 0$ for $j \neq i$; in that case, the parameter ψ_i is irrelevant and can be replaced by 0). For the moment, the transfer policy is assumed to be given; later we shall examine the effects of different transfer policies on the performance of the system.

When jobs are transferred in all directions, no queue can be analyzed independently of the others. An exact solution for the general model, even of the sort described in Section 4.3, is unattainable. We therefore seek an acceptable approximation.

Denote by $p_{i,n}$ the steady-state marginal probability that there are n jobs in queue i (including the one in service, if any). Let γ_i be the total arrival rate into queue i (it includes the external arrivals and the transfers from other queues). Since jobs complete service in queue i at the rate of $\mu_i(1 - p_{i,0})$, the rate at which jobs are transferred away from queue i is equal to $\gamma_i - \mu_i(1 - p_{i,0})$ (note that if $q_{i,i} = 1$, then $\gamma_i = \mu_i(1 - p_{i,0})$, and that jobs may be transferred even if they have already been transferred from another server). Hence, the internal traffic rate from queue i to queue j, $\gamma_{i,j}$, is given by

$$\gamma_{i,j} = [\gamma_i - \mu_i(1 - p_{i,0})]q_{i,j} \, . \tag{5.1}$$

Therefore, the total arrival rates, γ_i, satisfy the following set of traffic equations:

$$\gamma_i = \lambda_i + \sum_{j=1}^{N} \gamma_{j,i} = \lambda_i + \sum_{j=1}^{N} \gamma_j q_{j,i} - \sum_{j=1}^{N} \mu_j(1 - p_{j,0})q_{j,i} \, . \tag{5.2}$$

Unfortunately, we cannot use (5.2) to determine γ_i exactly. This is because the probabilities $p_{j,0}$ are unknown; they depend in a non-trivial way on all system parameters. However, an approximate solution can be obtained as follows:

1. Assume that the process of arrivals into queue i, merging external and internal arrivals, is Poisson. Make initial guesses for the rates γ_i and $\gamma_{i,j}$, choosing them to be underestimates; e.g. $\gamma_i = \lambda_i$, $\gamma_{i,j} = 0$, $(i, j = 1, 2, \ldots, N)$.

2. For each $i = 1, 2, \ldots, N$ do the following:

   (a) treating queue i in isolation, with arrival rate γ_i, service rate μ_i and reneging rate $\psi_i(1 - q_{i,i})$, compute $p_{i,0}$ according to (4.8) or (2.13);

   (b) for each $j = 1, 2, \ldots, N$, compute $\gamma_{i,j}$ according to (5.1) and update γ_j according to (5.2).

3. Iterate step 2 until the successive estimates are sufficiently close (e.g., until the sum of the absolute values of the differences, $|\gamma_i(next) - \gamma_i(last)|$, is less than some small ϵ).

The above solution will be referred to as the 'Poisson approximation'. Whenever the system is stable, the iterations have been observed to converge. While having no proof, we can offer the following intuitive argument to explain why this should be so: each application of step 2 tends to increase the internal traffic between queues, and hence the total arrival rates. Thus, the sequence of successive estimates is monotone, and since it is bounded in a stable system, it converges to the fixed point of equations

(5.2).

The computed values γ_i and $p_{i,0}$ yield, according to (4.10), estimates for the average number of jobs in queue i, L_i. In this general model, there are no expressions for the numbers of jobs in queue i that originated in queue j. The overall system performance will be measured by a cost function, C, which takes into account queue sizes and internal traffic:

$$C = \sum_{i=1}^{N} c_i L_i + \sum_{i=1}^{N} \sum_{j=1}^{N} c_{i,j} \gamma_{i,j} \, , \tag{5.3}$$

where the coefficients c_i and $c_{i,j}$ reflect the job holding costs in queue i per unit time, and the job transfer costs from queue i to queue j, respectively.

The reason for including the second term in (5.3) is as follows. If jobs can be transferred from queue i to queue j and also from queue j to queue i, and if those transfers do not incur costs, then the best time-out intervals in both queues are 0. Instantaneous transfers away from busy servers have the effect of creating a common queue for the two servers, which is the most efficient utilization of resources. However, in practice transfers *do* incur costs; the cost function should penalize not only long queues but also large numbers of transfers.

Having a computational procedure that determines C for a given set of parameters, one can address the problem of choosing the transfer policy Q, and the time-out rates ψ_i, so as to optimize the system performance. Some results along those lines are reported in the next section, where the performance of several heuristic policies are evaluated and compared.

5.3 Policies

In principle, if all parameters are known, one could search through the set of all feasible matrices Q (or rather a reasonably dense finite subset) and then through all values of ξ_i (again, a suitable finite subset), in order to find the best policy. We have done this for a model with 3 queues, and where applicable is plotted as the Optimal policy, as it determines the best of all possible policies. However, the brute force approach is generally impractical, not only because of the size of the search space but also because of the limited information available. Thus, it may be reasonable to assume that server i knows the speeds of the other servers (parameters μ_j), but not the corresponding arrival rates (λ_j), and hence defining heuristic transfer policies.

In general, the best transfer rates are determined by computing the cost function:

$$C = c_h \sum_{i=1}^{N} L_i + c_t \sum_{i=1}^{N} \sum_{j=1}^{N} \gamma_{i,j} \,. \tag{5.4}$$

where c_h is the holding cost per job, and c_t is the transfer cost, which we have determined should be $5 - 10$ times larger than c_h.

In practice, as the possible values of ψ_i are infinite, the cost function is evaluated for a set of values within a certain range, using the Poisson approximation method, described in Section 5.2, and choosing the rates that give the smallest cost. This is simplified by the observed behaviour of the cost as ψ is varied (see, for example, Figure 4.2, from which it is clear it is a convex function).

We have evaluated the performance of a few simple and easily implementable heuristic policies which do not require knowledge of arrival rates. Their definitions are as follows:

1. *No transfers*: The matrix Q is the unit matrix of size N. Alternatively, all transfer rates are 0.

2. *Uniform*: Jobs are transferred from queue i to queue j ($j \neq i$) with probability $1/(N-1)$.

3. *Speed-weighted*: Jobs are transferred from queue i to queue j ($j \neq i$) with probability proportional to μ_j (normalized so that the ith row-sum of Q is 1).

4. *Fastest other*: Number the queues in non-increasing order of service rates. Queues $2, 3, \ldots N$ send their transfers to queue 1; those from queue 1 go to queue 2.

5. *Next faster*: Number the queues in non-increasing order of service rates. Jobs from queue i are transferred to queue $i-1$ ($i > 1$); there are no transfers from queue 1.

6. *Equal load*: This policy does not employ time-outs, but achieves equal loads at all queues by transferring jobs at moments of arrival. More precisely, every job arriving into queue i is sent to queue j with probability $\mu_j/(\mu_1 + \mu_2 + \ldots + \mu_N)$; that decision involves an immediate transfer if $j \neq i$.

7. *Optimal*: As described in the discussion in this section.

In the following experiments, it is assumed that the holding costs at all queues are equal; the transfer costs between any two queues are also equal, but grater than the holding costs by a factor of 5. Thus, the performance of the system is measured by the cost function:

$$C = \sum_{i=1}^{N} L_i + 5 \sum_{i=1}^{N} \sum_{j=1}^{N} \gamma_{i,j} \, . \tag{5.5}$$

Under policies 1 and 6, the model consists of N independent M/M/1 queues.

5.4 Results

Figure 5.2 illustrates the benefit obtained by applying job timeouts and hence transfers in both directions between two queues. In this experiment, the external arrival rate into queue 1 is varied, while holding the arrival rate into queue 2 constant,

$\lambda_2 = 1.5$, as well as the service rate of each queue, $\mu_i = 1.8$. In this figure, the vertical bar at $\lambda_1 = 1.5$ represents the point at which the most highly loaded server changes from being server 2 (at $\lambda_1 < 1.5$) and becomes server 1 (at $\lambda_1 > 1.5$).

At low λ_1, there is little difference between only transferring in one direction, and transferring in both. This is due to the relatively large available service capacity, so when transfers of jobs are permitted from queue 1 to queue 2, very few jobs will timeout and transfer. However, as λ_1 becomes larger, the average queue length at queue 1 will increase, and so the optimal behaviour will be to transfer jobs between queues in both directions.

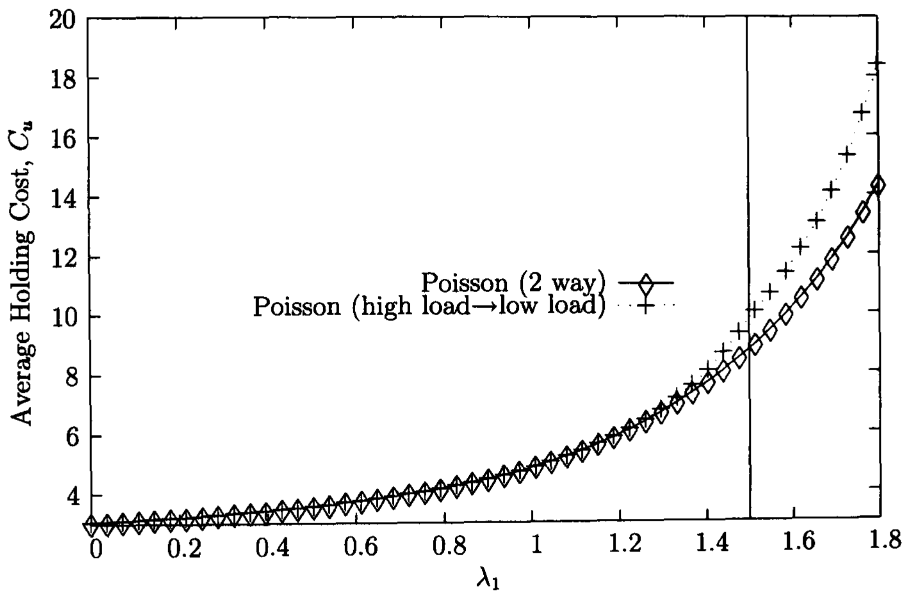

Figure 5.2: The benefits of two-way transfers

The next figure (5.3) illustrates the optimal values of ψ_i obtained using the Poisson approximation, with the same system parameters as before. As expected, when the external arrival rate into server 1,λ_1, is low, there are no transfers from server 1 to server 2, and the transfer rate from server 2 to server 1 is high. As λ_1 increases, the optimal transfer rate from server 2 decreases as the average queue length of server 1 increases.

When the external arrival rate into server 1 is the same as that into server 2, the optimal transfer rate from each server is the same, due to all parameters of each server being equal. When the overall system load is greater than ≈ 0.75, the transfer rate of each server is above zero.

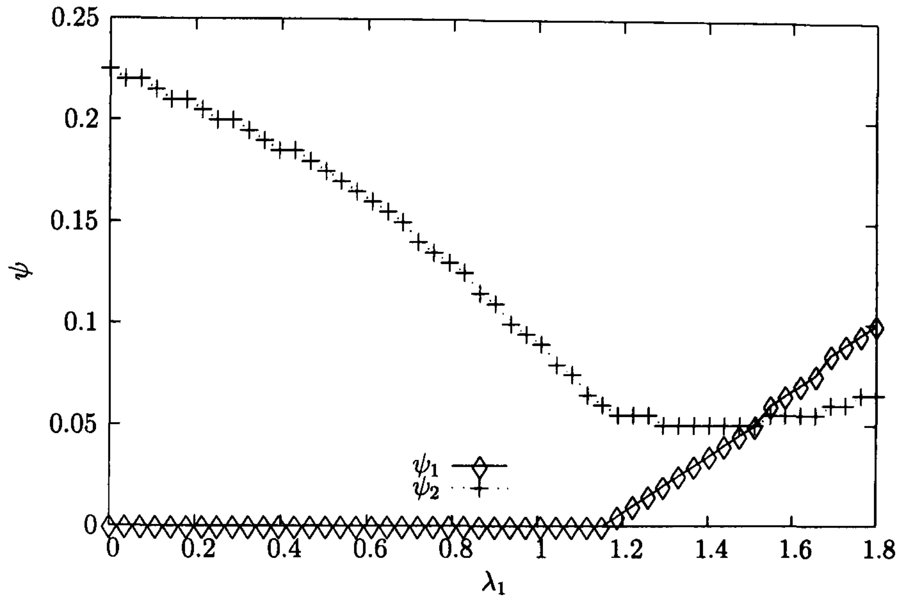

Figure 5.3: Optimal transfer rates

In Figure 5.4, the effect of varying the arrival rate into one server, while keeping the parameters of the remaining servers constant (there are 3 servers in total). The arrival rates into servers 2 and 3 are $\lambda_2 = 0.2$ and $\lambda_3 = 0.8$ and the arrival rates are $\mu_1 = 1.5$, $\mu_2 = 1.2$ and $\mu_3 = 1.0$.

As expected, the *No Reneging* policy performs badly across the full parameter range. In general, the *Equal Load* policy is far from *optimal*, as jobs are transferred

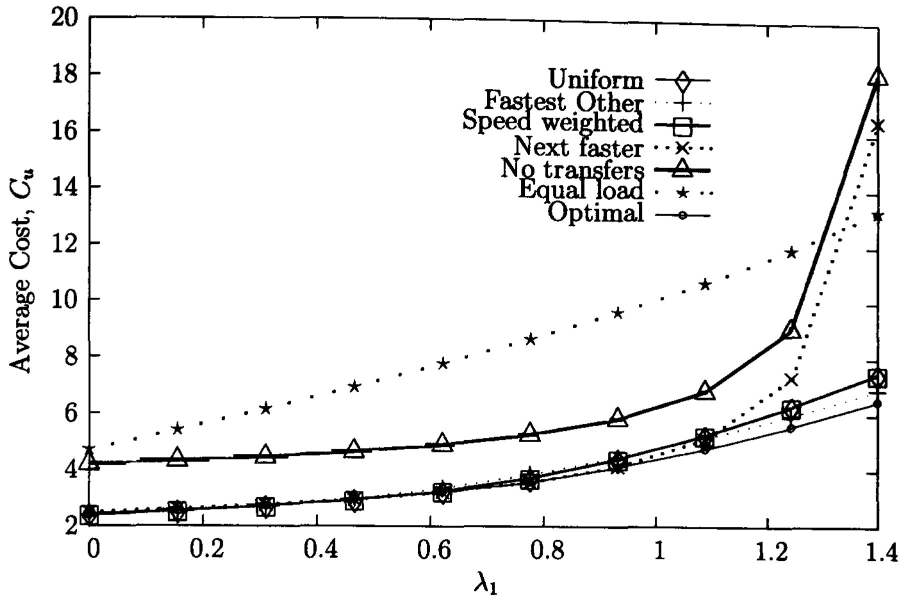

Figure 5.4: Effect of varying arrival rate

on arrival to ensure that the applied loads on each server is constant, even if there is free capacity at the server to which it originally arrived. At low values of λ_1, the *Just Faster* policy is close to *optimal.* However, once the arrival rate into server 1 is over 1, it rapidly becomes further from *optimal.* This is due to the policy behaviour, where jobs are transferred from server 1 $\rightarrow$ 2, but jobs can be also be transferred from 3 $\rightarrow$ 1, and also 2 $\rightarrow$ 1 (In fact, it starts to become worse than *Optimal* at ≈ 0.5, but the effect is small until ≈ 1).

The *All Others* and *Probabilistic* policies are equally close to *optimal* across the whole range of the experiment, and are both only slightly outperformed by the *Fastest Other* policy which is about 50% closer to *optimal.*

Figure 5.5 shows the effect of increasing the number of servers in each of two groups. The arrival rates into group 1 servers, $\lambda_1 = 1.0$, and the arrival rate into group 2 servers, $\lambda_2 = 1.5$, are kept constant, as are the service rates of both groups,

$\mu_1 = 1.4$ and $\mu_2 = 1.6$.

When $N = 2$, the policies split into two, with the *Equal Load*, *No Reneging* and *Just Faster* policies in the worst performing group, with *Equal Load* performing just better than the other two, which remain together across the whole experiment. As N increases, the *Equal Load* policy becomes gradually much worse performing than the others in its group, due to the inability to take advantage of variations in queue length, after the arrival of the jobs.

In the lower group of policies, the *Fastest Other* policy gradually worsens, compared to the rest of the group, but never to the extent that it meets the worse performing group of policies. The *All Others* and *Probabilistic* policies produce the same average costs over the whole range of the experiment, and are overall the best performing.

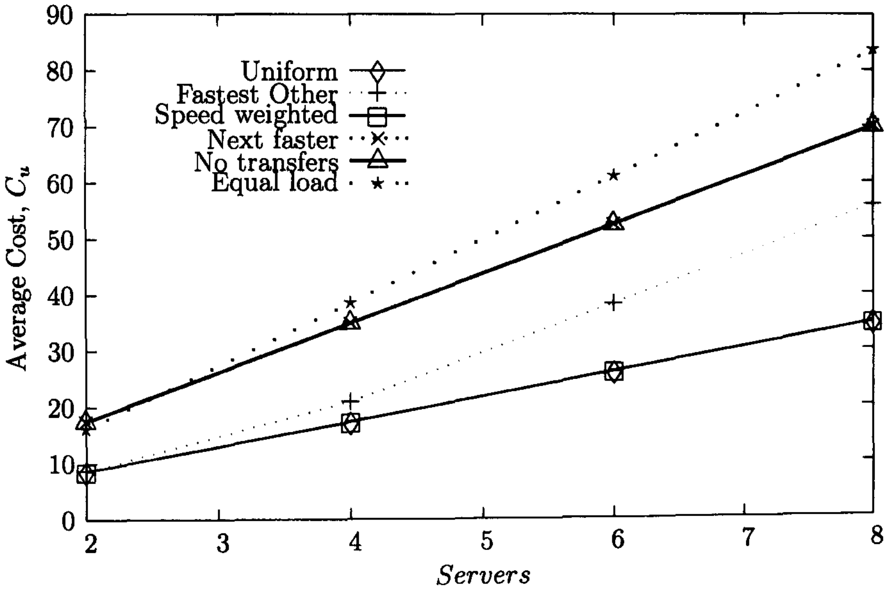

Figure 5.5: The effect of increasing group size

For all tested examples for $N > 3$, the same general behaviour was observed, with

Speed Weighted and *Uniform* both performing well, and increasingly better than all other policies. When the number of servers is fixed at a large ($N > 3$) fixed value, and one of the system parameters is varied as above, the *Fastest Other* policy performs worse than the noted best pair of policies. This is due to their ability to transfer to any other server, whereas the *Fastest Other* policy only ever transfers jobs to one of two servers.

5.5 Conclusions

For systems of servers with general transfers, an iterative approximation approach was used to find the cost of various transfer policies, according to a cost function which includes both holding cost and cost of job transfers. All these policies were simple in terms both of their description and implementation and several were found to perform well in comparison to the best policy found by a brute force search approach.

It was observed that policies which performed well over a broad range of system parameters and sizes had the general characteristic of transferring jobs to any other server, which prevents any one queue from growing large while others have fewer or no jobs waiting.

Chapter 6

Unreliable Servers with Job Transfers

6.1 Motivation

In service provisioning systems such as a Computing Grid, it is likely that servers will undergo periods where they are unavailable to service jobs for the system. In the case of transfer-based systems as discussed over the last two chapters, if the expected duration of the unavailable period is low it may allow a job to complete service earlier (or give it a smaller expected cost) if it waits for the period to expire rather than transferring to another server in the system.

The system in this chapter is of this form. Once again, the distribution of jobs in the system is evaluated using an iterative approximation approach, and then a cost function is evaluated to allow transfer policies to be compared.

6.2 Model Definition

The system consists of N servers, each with a separate queue. Jobs arrive *externally* into queues i in an independent Poisson stream with rate λ_i $i = 1, 2, \ldots, N$. The service times at server i are independent, exponentially distributed random variables with mean $1/\mu_i$. Each server goes through alternating independent periods of being available and unavailable, or operative and inoperative. We shall call the end points of those periods 'breakdowns' and 'repairs', respectively, although in practice the main reason for the occurrence of periods of unavailability is that the server is occupied with other, higher priority tasks. The operative and inoperative periods for server i are distributed exponentially with means $1/\xi_i$ and $1/\eta_i$, respectively.

Whenever a queue is not empty and its server is available, one of the jobs in it is being served. Services interrupted by breakdowns are eventually resumed from the point of interruption, but not necessarily on the same server. Each job in queue i which is not being served is assigned an independent time-out period which is distributed exponentially with mean $1/\psi_{i,0}$ if the server is inoperative, and mean $1/\psi_{i,1}$ if the server is operative. Any job whose time-out period expires before it reaches the server is instantaneously transferred to another queue, determined by the transfer policy adopted (e.g., the destination queue may be chosen at random). A three-server system is illustrated in Figure 6.1.

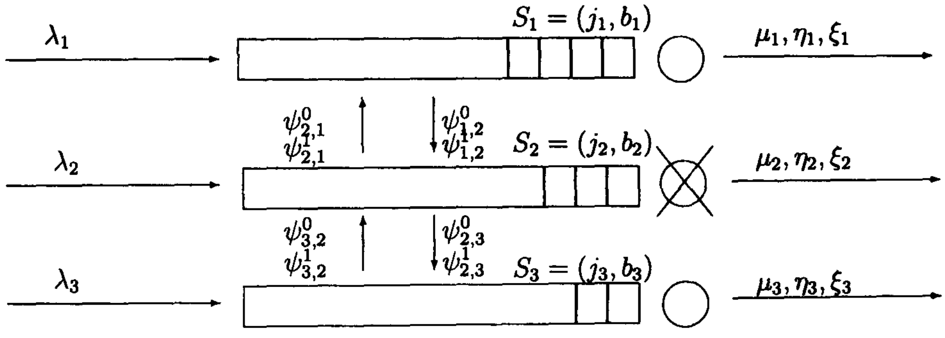

Figure 6.1: Unreliable Servers with Job Transfers

Note that if there are $j > 0$ jobs in queue i, then the instantaneous rate of transfers from that queue is $(j-1)\psi_{i,1}$ if the server is operative (since one of the jobs is being served), and $j\psi_{i,0}$ if the server is inoperative. Also, if the server changes state from operative to inoperative or vice versa, then all waiting jobs have their time-out intervals re-sampled with the new mean.

Denote by L_i the steady state average numbers of jobs in queue i. Let also $\beta_{i,j}$ be the steady state average number of transfers from queue i to queue j per unit time.

Suppose that it costs c_i to keep a job in queue i per unit time, and it costs $c_{i,j}$ to transfer a job from queue i to queue j. Then, the total average cost per unit time in the steady state is given by

$$C = \sum_{i=1}^{N} c_i L_i + \sum_{i,j=1}^{N} c_{i,j} \beta_{i,j} . \tag{6.1}$$

The ergodicity condition for this system is, in general, unknown. What we can say is that server i is operative for a fraction $\eta_i/(\xi_i + \eta_i)$ of the time. Hence, the total available service capacity, μ, is equal to

$$\mu = \sum_{i=1}^{N} \frac{\mu_i \eta_i}{\xi_i + \eta_i} . \tag{6.2}$$

Clearly, the total external arrival rate must be lower than the available service capacity:

$$\sum_{i=1}^{N} \lambda_i < \mu . \tag{6.3}$$

However, although this condition is necessary for stability, it may not be sufficient. The load-balancing mechanism using job transfers does not preclude situations where some servers are operative and idle, while others have more than one job in their queues.

The performance of the system is controlled by means of the time-out intervals, and also the job transfer policy. The objective is to choose those parameters so as to minimize the cost function (6.1). In order to do that, we need to provide solution methods for determining the performance measures L_i and $\beta_{i,j}$. That will be the aim of sections 6.3 and 6.4.

6.3 An isolated queue

Consider a single queue with Poisson input, intermittently available server, and time-outs resulting in departures (reneging). Omitting the queue index, the parameters are λ (arrival rate), μ (service rate), ξ (breakdown rate), η (repair rate), ψ_0 (reneging rate during inoperative periods) and ψ_1 (reneging rate during operative periods).

The system state is described by a pair of integers, (i, j), where $j = 0, 1, \ldots$ is the number of jobs present and $i = 0$ if the server is inoperative, $i = 1$ if it is operative. Denote by $p_{i,j}$ the equilibrium probability of state (i, j). These probabilities satisfy the following set of balance equations.

$$(\lambda + j\psi_0 + \eta)p_{0,j} = \lambda p_{0,j-1} + (j+1)\psi_0 p_{0,j+1} + \xi p_{1,j} , \tag{6.4}$$

$$[\lambda + \mu\delta(j > 0) + (j-1)\psi_1 + \xi]p_{1,j} = \lambda p_{1,j-1} + (\mu + j\psi_1)p_{1,j+1} + \eta p_{0,j} , \tag{6.5}$$

where all probabilities with a negative index are 0 by definition; the Boolean indicator function, $\delta(B)$, is 1 if B is true, 0 otherwise.

It is convenient to introduce the generating functions

$$g_0(z) = \sum_{j=0}^{\infty} p_{0,j} z^j \ ; \ g_1(z) = \sum_{j=0}^{\infty} p_{1,j} z^j . \tag{6.6}$$

Multiplying (6.4) by z^j and summing over all j, those balance equations can be re-written as

$$g_0'(z) = \frac{1}{\psi_0}\left[\lambda + \frac{\eta}{1-z}\right] g_0(z) - \frac{\xi g_1(z)}{\psi_0(1-z)} . \tag{6.7}$$

Similarly, equations (6.5) yield

$$g_1'(z) = \frac{1}{\psi_1}\left[\lambda + \frac{\xi}{1-z} - \frac{\mu - \psi_1}{z}\right] g_1(z) + \frac{\mu - \psi_1}{\psi_1 z} p_{1,0} - \frac{\eta g_0(z)}{\psi_1(1-z)} . \tag{6.8}$$

This is a set of two first order linear differential equations with two unknown functions, involving also the probability, $p_{1,0} = g_1(0)$, that the server is operative and idle. The following are some simple consequences from these equations.

In order that $g_0'(1)$ and $g_1'(1)$ be finite, we must have $\eta g_0(1) = \xi g_1(1)$. This, together with the normalizing equation, $g_0(1) + g_1(1) = 1$, provides the marginal probabilities of the server being inoperative and operative (also obtainable directly from the nature of breakdowns and repairs)

$$g_0(1) = \frac{\xi}{\xi + \eta} \; ; \; g_1(1) = \frac{\eta}{\xi + \eta} \, . \tag{6.9}$$

Next, setting $z = 1$ in (6.7) and (6.8) and applying L'Hopital's rule, we obtain

$$g_0'(1) = \frac{1}{\psi_0}[\lambda\alpha + \xi g_0'(1) - \eta g_1'(1)] \, , \tag{6.10}$$

$$g_1'(1) = \frac{1}{\psi_1}[(\lambda - \mu + \psi_1)(1 - \alpha) + (\mu - \psi_1)p_{1,0} + \eta g_0'(1) - \xi g_1'(1)] \, , \tag{6.11}$$

where $\alpha = g_0(1) = \xi/(\xi + \eta)$ is the probability that the server is inoperative.

These equations can be solved to yield an expression for the average number of jobs present, $L = g_0'(1) + g_1'(1)$, in terms of $p_{1,0}$.

$$L = \frac{(\xi + \eta + \psi_0)[\lambda - (\mu - \psi_1)(1 - \alpha - p_{1,0})] - \alpha\lambda(\psi_0 - \psi_1)}{\psi_0\psi_1 + \xi\psi_0 + \eta\psi_1} \, . \tag{6.12}$$

The other performance measure for the isolated queue is the average number of jobs that renege per unit time, denoted by β. Since the reneging jobs are those arrivals that do not complete service, and since jobs are completed at rate μ when the server is operative and not idle, we can write

$$\beta = \lambda - \mu(1 - \alpha - p_{1,0}) \, . \tag{6.13}$$

It now remains to determine the probability $p_{1,0}$. There are several ways of approaching this task.

6.3.1 Integrals and iterations

Treating the function $g_1(z)$ which appears in the right-hand side of (6.7) as known, and applying the known formula for the solution of the first-order linear differential equation (e.g., see [18]), we can write, after some manipulation,

$$g_0(z) = \frac{\xi}{\psi_0} e^{\frac{\lambda}{\psi_0} z} (1-z)^{-\frac{\eta}{\psi_0}} \int_z^1 e^{-\frac{\lambda}{\psi_0} x} (1-x)^{\frac{\eta}{\psi_0}-1} g_1(x) dx \, . \tag{6.14}$$

Similarly, by solving (6.8), $g_1(z)$ can be expressed in terms of an integral involving $g_0(z)$ and $p_{1,0}$.

$$g_1(z) = e^{\frac{\lambda}{\psi_1} z} (1-z)^{-\frac{\xi}{\psi_1}} z^{1-\frac{\mu}{\psi_1}} \left[\frac{\eta}{\psi_1} \int_z^1 e^{-\frac{\lambda}{\psi_1} x} (1-x)^{\frac{\xi}{\psi_1}-1} x^{\frac{\mu}{\psi_1}-1} g_0(x) dx \right.$$

$$\left. - \, p_{1,0} \frac{\mu - \psi_1}{\psi_1} \int_z^1 e^{-\frac{\lambda}{\psi_1} x} (1-x)^{\frac{\xi}{\psi_1}} x^{\frac{\mu}{\psi_1}-2} dx \right] . \tag{6.15}$$

Assume that $\psi_1 < \mu$ (this is the case of practical interest, since the time-out interval is unlikely to be smaller than a service time when the server is operative). Then the finiteness of $g_1(0)$ implies that the square brackets in the right-hand side of (6.15) must vanish at $z = 0$. Hence,

$$p_{1,0} = \frac{\eta \int_0^1 e^{-\frac{\lambda}{\psi_1} x} (1-x)^{\frac{\xi}{\psi_1}-1} x^{\frac{\mu}{\psi_1}-1} g_0(x) dx}{(\mu - \psi_1) \int_0^1 e^{-\frac{\lambda}{\psi_1} x} (1-x)^{\frac{\xi}{\psi_1}} x^{\frac{\mu}{\psi_1}-2} dx} \, . \tag{6.16}$$

The last three equations suggest an iterative scheme for computing the solution. Start by making an initial guess for the function $g_0(z)$. For example, treat the queue during inoperative periods as an M/M/∞ queue with parameters λ and ψ_0, and use $g_0(z) = \alpha \exp(\lambda(z-1)/\psi_0)$. This, together with (6.16), provides an initial estimate for $p_{1,0}$ and also, using (6.15), a first estimate for $g_1(z)$. Equation (6.14) then yields a second estimate for $g_0(z)$, and so on, until some termination criterion is satisfied.

Although the above scheme is viable and has been implemented, we do not really recommend it as a solution method. It suffers from numerical instability problems, particularly in the neighborhoods of $z = 0$ and $z = 1$, which require careful handling.

6.3.2 Long operative/inoperative periods

In real systems the periods of availability and unavailability tend to be much larger than the interarrival and service times. It is therefore worth considering the asymptotic performance of the queue as $\xi \to 0$ and $\eta \to 0$, while keeping their ratio constant (i.e., the steady state probability that the server is inoperative, α, is fixed).

Setting $\xi = \eta = 0$ in (6.7) turns that equation into

$$g_0'(z) = \frac{\lambda}{\psi_0} g_0(z) . \tag{6.17}$$

The solution which satisfies $g_0(1) = \alpha$ is, as expected, the normalized generating function of the corresponding M/M/∞ queue.

$$g_0(z) = \alpha e^{\frac{\lambda}{\psi_0}(z-1)} . \tag{6.18}$$

Similarly, the limiting equation (6.8) is

$$g_1'(z) = \frac{1}{\psi_1}\left[\lambda - \frac{\mu - \psi_1}{z}\right] g_1(z) + \frac{\mu - \psi_1}{\psi_1 z} p_{1,0} . \tag{6.19}$$

Its solution satisfying $g_1(1) = 1 - \alpha$ is given by

$$g_1(z) = e^{\frac{\lambda}{\psi_1}z} z^{1-\frac{\mu}{\psi_1}} \left[(1-\alpha)e^{-\frac{\lambda}{\psi_1}} - p_{1,0}\frac{\mu - \psi_1}{\psi_1}\int_z^1 e^{-\frac{\lambda}{\psi_1}x} x^{\frac{\mu}{\psi_1}-2} dx\right] . \tag{6.20}$$

Again assuming that $\psi_1 < \mu$, the square brackets in the right-hand side of (6.20) must vanish at $z = 0$. This leads to an expression for $p_{1,0}$ which, after integrating by parts and changing variables, can be written as

$$p_{1,0} = (1-\alpha)\left[1 + e^{\frac{\lambda}{\psi_1}}\left(\frac{\psi_1}{\lambda}\right)^{\frac{\mu}{\psi_1}-1} \gamma(\frac{\lambda}{\psi_1}, \frac{\mu}{\psi_1})\right]^{-1} , \tag{6.21}$$

where $\gamma(x, y)$ is the incomplete gamma function:

$$\gamma(x, y) = \int_0^x e^{-u} u^{y-1} du .$$

Equation (6.21) provides a simple and robust approximation for the performance measures when the periods of availability and unavailability are long.

6.3.3 Exact solution when $\psi_1 = 0$

Intuitively, one might expect that setting $\psi_1 = 0$, i.e. not transferring jobs away from a queue when its server is operative, would be quite a good policy. Indeed, that turns out to be the case (see Section 6.5). Hence, that special case is of interest in its own right. Moreover, the simplification involved is enough to enable us to obtain an exact solution in closed form.

The differential equation (6.7) for $g_0(z)$ remains unchanged. However, when $\psi_1 = 0$, the balance equations (6.5) lead to an algebraic, rather than differential, equation for $g_1(z)$:

$$g_1(z) = \frac{\eta z}{d(z)} g_0(z) - \frac{\mu(1-z)}{d(z)} p_{1,0} , \tag{6.22}$$

where

$$d(z) = (\lambda z - \mu)(1-z) + \xi z .$$

Substituting (6.22) into (6.7) and simplifying, we get

$$g_0'(z) = \left[\frac{\lambda}{\psi_0} + \frac{\eta(\lambda z - \mu)}{\psi_0 d(z)} \right] g_0(z) + \frac{\xi\mu}{\psi_0 d(z)} p_{1,0} . \tag{6.23}$$

Note that apart from the function $g_0(z)$, this equation involves only the constant $p_{1,0}$, not another unknown function.

Now, since the quadratic polynomial $d(z)$ is negative at $z = 0$, positive at $z = 1$ and negative at $z = \infty$, its two zeros, z_1 and z_2, are real and satisfy $0 < z_1 < 1$ and $1 < z_2$. Writing $d(z) = \lambda(z - z_1)(z_2 - z)$ and decomposing the second term in the square brackets in (6.23) into elementary fractions, that equation can be re-written as

$$g_0'(z) = \left[\frac{\lambda}{\psi_0} - \frac{a\xi\eta}{\lambda\psi_0(z - z_1)} + \frac{b\xi\eta}{\lambda\psi_0(z_2 - z)} \right] g_0(z) + \frac{\xi\mu}{\psi_0 d(z)} p_{1,0} , \tag{6.24}$$

where

$$a = \frac{z_1}{(1-z_1)(z_2-z_1)} \; ; \; b = \frac{z_2}{(z_2-1)(z_2-z_1)} \, .$$

The solution of (6.24) is given by

$$\begin{aligned} g_0(z) &= e^{\frac{\lambda}{\psi_0}z}(z-z_1)^{-\frac{a\xi\eta}{\lambda\psi_0}}(z_2-z)^{-\frac{b\xi\eta}{\lambda\psi_0}} \\ & \left[C - p_{1,0}\frac{\xi\mu}{\lambda\psi_0}\int_z^1 e^{-\frac{\lambda}{\psi_0}x}(x-z_1)^{\frac{a\xi\eta}{\lambda\psi_0}-1}(z_2-x)^{\frac{b\xi\eta}{\lambda\psi_0}-1}dx\right] , \end{aligned} \qquad (6.25)$$

where C is determined by the normalization $g_0(1) = \alpha$:

$$C = \alpha e^{-\frac{\lambda}{\psi_0}}(1-z_1)^{\frac{a\xi\eta}{\lambda\psi_0}}(z_2-1)^{\frac{b\xi\eta}{\lambda\psi_0}} \, .$$

The desired expression for $p_{1,0}$ is provided by remarking that $a > 0$, and $g_0(z_1)$ is finite. Hence, the square bracket in the right-hand side of (6.25) must vanish at $z = z_1$, which yields

$$p_{1,0} = C\frac{\lambda\psi_0}{\xi\mu}\left[\int_{z_1}^1 e^{-\frac{\lambda}{\psi_0}x}(x-z_1)^{\frac{a\xi\eta}{\lambda\psi_0}-1}(z_2-x)^{\frac{b\xi\eta}{\lambda\psi_0}-1}dx\right]^{-1} . \qquad (6.26)$$

6.3.4 Truncated state space

The general model of the isolated queue, when ξ and η are not necessarily small and ψ_1 is not 0, can also be tackled by truncating the state space and solving numerically the resulting finite set of linear balance and normalizing equations. This can be done without significant loss of accuracy.

Let $\psi > 0$ be the smaller of ψ_0 and ψ_1: $\psi = \min(\psi_0, \psi_1)$. If the queue is replaced by one where the server is unavailable all the time and the time-out parameter is ψ, then the departure rate would decrease and the tail of the queue size distribution would become thicker. Moreover, the modified queue behaves like an M/M/∞ queue with parameters λ and ψ.

Choose an arbitrary error bound, $\epsilon > 0$, and find an integer m such that

$$\left(\frac{\lambda}{\psi}\right)^{m+1} \frac{1}{(m+1)!} < \epsilon \,. \tag{6.27}$$

That is always possible. Then, the above argument shows that if the original queue is truncated at threshold m, i.e. new arrivals are not allowed to join it when there are m jobs present, the sum of the probabilities of the neglected states is less than ϵ.

We have found that this approach is in fact efficient and easily implementable, using either direct or iterative solution techniques.

6.4 Approximate solution for N queues

Let us now return to the general model of Section 6.2, consisting of N queues with different parameters. Any job whose time-out period expires before it reaches the server in queue i, is instantaneously transferred to queue j with probability $q_{i,j}$ (that probability does not depend on the operative state of server j because the latter is assumed unknown).

A matrix of transfer probabilities, $Q = \{q_{i,j}\}_{i,j=1}^{N}$, all of whose row sums are equal to 1, defines a 'transfer policy' for the system. We consider policies which transfer from queue i either all jobs that time out in it, or none of them. In other words, either $q_{i,i} = 0$ (and the other probabilities in row i add up to 1), or $q_{i,i} = 1$ (and $q_{i,j} = 0$ for $j \neq i$; in that case, the parameters ψ_0 and ψ_1 are irrelevant. For the moment, the transfer policy is assumed to be given; later we shall examine the effects of different transfer policies on the performance of the system.

When jobs are transferred in all directions, the queues are coupled and their joint and marginal distributions are intractable. We therefore seek an acceptable approximation.

Denote by γ_i be the total arrival rate into queue i in the steady state, under policy Q (that rate includes the external arrivals and the transfers from other queues). Let β_i be the rate at which jobs are transferred away from queue i. The transfer rate from queue i to queue j is then equal to $\beta_i q_{i,j}$, and we can write a set of traffic equations for γ_i.

$$\gamma_i = \lambda_i + \sum_{j=1}^{N} \beta_j q_{j,i} \, . \tag{6.28}$$

Unfortunately, the quantities β_i are unknown; they depend in a non-trivial way on all system parameters. However, an approximate solution can be obtained as follows:

1. Assume that the process of arrivals into queue i, merging external and internal arrivals, is Poisson. Make initial guesses for the rates γ_i, choosing them to be underestimates; e.g. $\gamma_i = \lambda_i$, $i = 1, 2, \dots, N$.

2. For each $i = 1, 2, \dots, N$, treating queue i in isolation and applying one of the solution methods described in section 3, determine β_i according to (6.13) (with λ replaced by γ_i).

3. Using (6.28), compute a new estimate for γ_i.

4. Iterate from step 2 until the successive estimates are sufficiently close (e.g., until the sum of the absolute values of the differences, $|\gamma_i(next) - \gamma_i(last)|$, is less than some small ϵ).

5. Determine the cost of the system, based on the average numbers of jobs in queue i, L_i, (computed according to (6.12)), and the average transfer rates, β_i.

The above solution will be referred to as the 'Poisson approximation'. Whenever the system is stable, the iterations have been observed to converge. While having no proof, we can offer the following intuitive argument to explain why this should be so: each application of steps 2 and 3 tends to increase the internal traffic between queues, and hence the total arrival rates. Thus, the sequence of successive estimates is monotone, and since it is bounded in a stable system, it converges to the fixed point of equations (6.28).

Having a computational procedure that determines the objective function for a given set of parameters, one can address the problem of choosing the transfer policy Q, and the time-out rates, so as to optimize the system performance. When N is small, one could search through the set of all feasible matrices Q (or rather a reasonably dense finite subset), and through all values of ψ_0 and ψ_1 for each queue (again, a suitable finite subset), in order to find the best configuration.

However, the brute force approach is generally impractical, not only because of the size of the search space but also because of the limited information available. Thus, it may be reasonable to assume that server i knows the speeds of the other servers (parameters μ_j), or their *effective* speeds (parameters $\mu_j^* = \mu_j\eta_j/(\xi_j + \eta_j)$), but not the corresponding arrival rates (λ_j). It is therefore worth introducing and evaluating some heuristic policies that one might consider implementing. Some of the following simple heuristics assume knowledge of effective service rates; none rely on knowing the arrival rates or the current states of servers and queues.

1. *No transfers*: The matrix Q is the unit matrix of size N. Alternatively, all transfer rates are 0.

2. *Uniform*: Jobs are transferred from queue i to queue j ($j \neq i$) with probability $1/(N-1)$.

3. *Speed-weighted*: Jobs are transferred from queue i to queue j ($j \neq i$) with probability proportional to μ_j^* (so that the ith row of Q adds up to 1).

4. *Equal load*: This policy does not employ time-outs, but achieves equal loads at all queues by transferring jobs at moments of arrival. More precisely, every job arriving into any queue, i, is sent to queue j with probability $\mu_j^*/(\mu_1^* + \mu_2^* + \ldots + \mu_N^*)$; that decision involves an immediate transfer if $j \neq i$.

5. *Fastest other*: Number the queues in decreasing order of effective service rates (parameters μ_i^*). From queue 1, jobs are transferred to queue 2; from any other queue they are transferred to queue 1.

6. *Round-robin*: Using the same numbering as in policy 5, transfer jobs from queue i to queue $i+1$ if $i < N$, and from queue N to queue 1.

Under policies 1 and 4, the model consists of N independent M/M/1 queues. For the other policies, the cost function would normally be computed by means of the Poisson approximation.

6.5 Numerical results

Several numerical and simulation experiments were carried out, attempting to answer the following questions:

1. Is the Poisson approximation acceptable?
2. How should one choose the time-out rates, $\psi_{i,0}$ and $\psi_{i,1}$, at queue i?
3. What gains in performance can be expected by using good choices of $\psi_{i,0}$ and $\psi_{i,1}$ as opposed to bad ones?
4. How do the heuristic policies compare to each other and to the optimal policy (when the latter can be computed)?

In all cases, the holding costs at all queues are assumed equal. The transfer costs are also equal, but are greater than the holding costs by a factor of 10. Thus, the coefficients of the cost function are $c_i = 1$, $c_{i,j} = 10$.

The first experiment examines the way performance is influenced by changing time-out parameters, in the context of a system with two queues. At the same time, the results provided by the Poisson approximation are compared with those obtained from simulations.

The two service rates are equal, as are the average operative and inoperative intervals. Each server is available 50% of the time. The arrival rates are chosen so that both queues are quite heavily loaded; the external offered load at queue 1 is 0.7, while that at queue 2 is 0.8. The average time-out intervals at queue 2 do not depend on the state of the server and are fixed at just over two service times: $\psi_{2,0} = \psi_{2,1} = 0.4$.

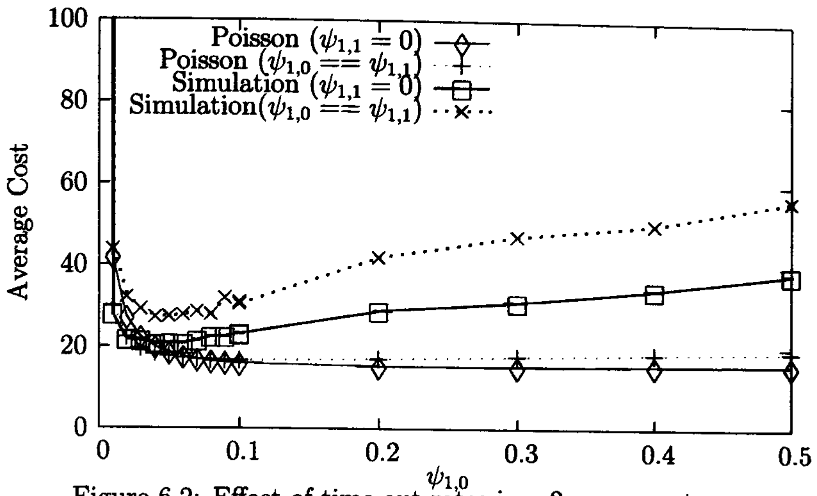

Figure 6.2: Effect of time-out rates in a 2-queue system
$\lambda_1 = 0.385$, $\lambda_2 = 0.44$, $\mu_i = 1.1$, $\xi_i = \eta_i = 0.1$, $\psi_{2,0} = \psi_{2,1} = 0.4$

In Figure 6.2, the average cost is plotted against $\psi_{1,0}$, the time-out rate in queue 1 when the server is inoperative. The rate $\psi_{1,1}$ is either equal to $\psi_{1,0}$ (state-independent timeouts), or is 0. The following remarks are suggested by these results.

- There is an optimal value for the time-out parameter. When $\psi_{1,0}$ is too small, the holding costs in queue 1 are dominant; that queue becomes saturated due to transfers from queue 2. As $\psi_{1,0}$ increases, the holding costs in queue 1 decrease, while those in queue 2, and the costs due to transfers, increase. However, the latter effects are less pronounced.

- The Poisson approximation underestimates the true costs. This is not really surprising, because the real arrival process has a higher variance than is implied by the Poisson assumption (particularly when $\psi_{1,0}$ and $\psi_{1,1}$ are different).

- The approximation overestimates the optimal value of the time-out parameter, but not by much. This is a consequence of the cost underestimation.

- Lower costs can be achieved by using state-dependent time-out rates than state-independent ones. In particular, it is better not to transfer jobs from a queue during operative periods.

Intuitively, the best time-out policy should depend on the lengths of operative and inoperative intervals. This is illustrated for a two-queue system in Figure 6.3, where the average operative periods at the two servers are equal, and are also equal to the average inoperative periods. Those lengths are increased and the costs of the following three policies are plotted:

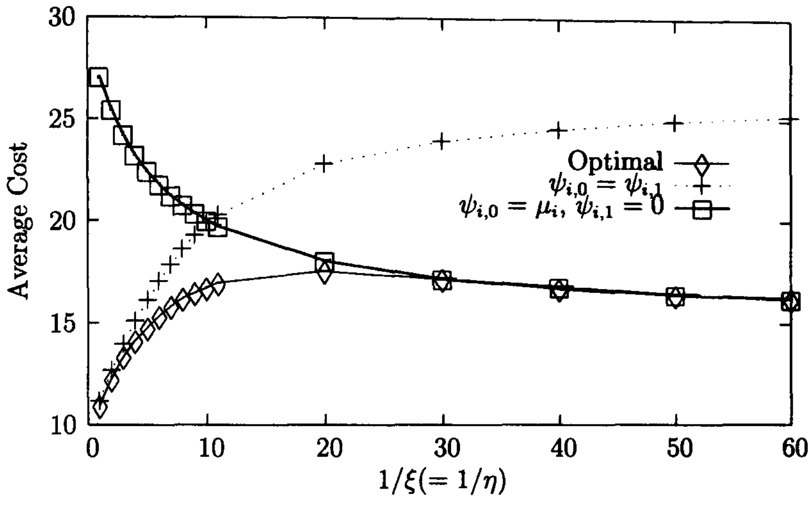

Figure 6.3: Increasing operative/inoperative periods
$\lambda_1 = 0.495$, $\lambda_2 = 0.44$, $\mu_i = 1.1$

(a) State-independent time-outs; the values of $\psi_{i,0} = \psi_{i,1}$ $(i = 1, 2)$ are chosen optimally by means of a search.

(b) No time-outs during operative periods ($\psi_{i,1} = 0$), quickly clear the queue when the server breaks down (large value for $\psi_{i,0}$, here chosen equal to μ_i).

(c) Optimal time-outs; the values of $\psi_{i,0}$ and $\psi_{i,1}$ $(i = 1, 2)$ are chosen by means of an exhaustive search.

All costs are computed using the Poisson approximation. This will be the case from now on.

Figure 6.3 shows that the state-independent time-out policy is almost optimal when the operative/inoperative intervals are short, but becomes very poor when those intervals are large. Conversely, policy (b) performs badly when the operative/inoperative intervals are short, but becomes almost optimal when the intervals are large. Since the latter case is most likely to occur in practice, the policy that does not transfer jobs during operative periods and quickly clears the queue after a breakdown, is a good candidate for adoption.

In the third experiment, the same 2-queue system is subject to an increasing offered load at queue 1 (increasing λ_1), while the other parameters are kept fixed. The operative and inoperative intervals are quite large. Figure 6.4 shows the costs achieved by the three policies (a), (b) and (c). These results confirm the near-optimality of policy (b), and emphasize the increasingly higher costs of state-independent time-outs relative to the state-dependent ones.

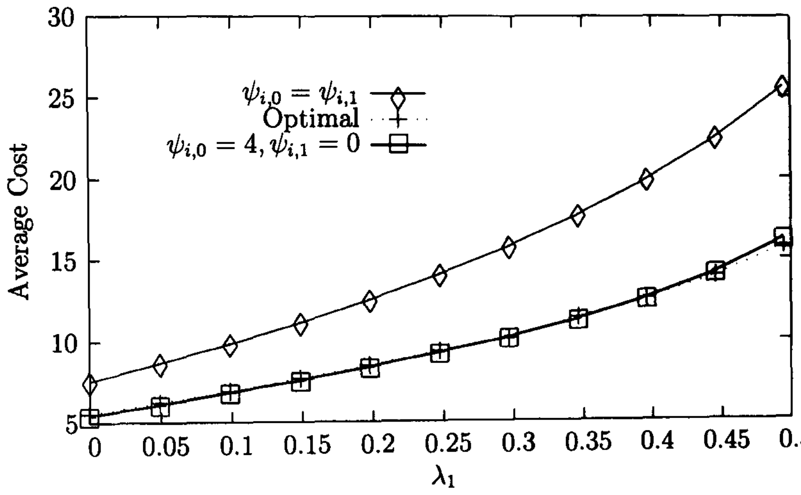

Figure 6.4: Increasing load at queue 1
$\lambda_2 = 0.44$, $\mu_1 = \mu_2 = 1.1$; $\xi_i = \eta_i = 0.01$

The last two experiments concern systems with more than two queues. They address the issue of transfer policy (i.e., where should one send a timed-out job), by evaluating and comparing the costs of the six heuristics defined in the last section.

In Figure 6.5, the model consists of three queues with different arrival and service parameters. The arrival rate at queue 1 is increased, all other parameters remain fixed. The time-out parameters at all queues are $\psi_{i,1} = 0$, $\psi_{i,1} = \mu_i$. The breakdown and repair rates are equal, so the availability of all servers is 50%.

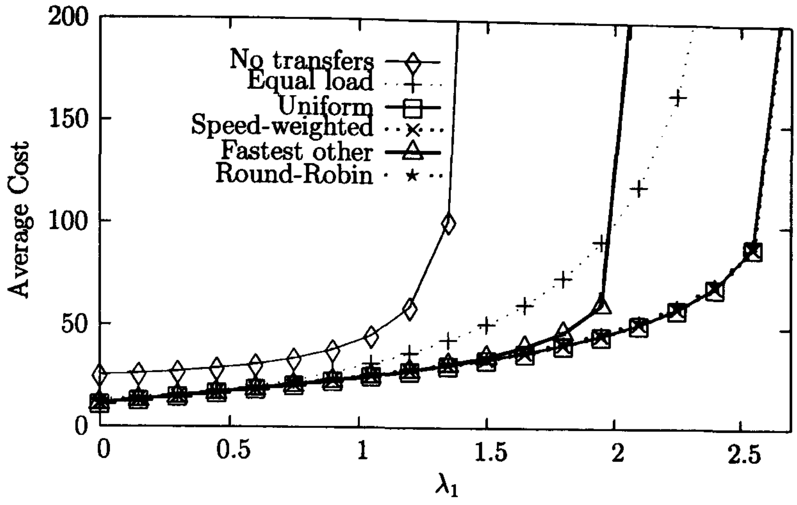

Figure 6.5: Different transfer policies in a system with 3 queues $\lambda_2 = 0.2$, $\lambda_3 = 0.8$ $\mu_1 = 3$, $\mu_2 = 2.4$, $\mu_3 = 2$, $\xi_i = \eta_i = 0.1$

As may be expected, the *No transfer* policy has the worst performance. It incurs large holding costs because queues grow during inoperative periods. Also, queue 1 becomes unstable at $\lambda_1 = 1.5$. The *Equal load* policy avoids early instability but it, too, incurs high holding costs during inoperative periods; in addition, its transfer costs are high because many jobs are transferred on arrival. The *Fastest other* policy performs well for low values of λ_1 but as the latter increases the cost escalates sharply. This is because both queues 2 and 3 send their transfers to queue 1, eventually saturating it.

The three policies *Uniform*, *Round-Robin* and *Speed-weighted*, perform well throughout the range of λ_1 values; There is little to choose between them, although *Speed-weighted* is slightly better than the other two. We conjecture that these policies are close to optimal.

In the final experiment, the number of queues is even and increases. Half of the queues have higher arrival rates and faster servers than the other half. The time-out parameters are the same as before. Again, all servers are 50% available.

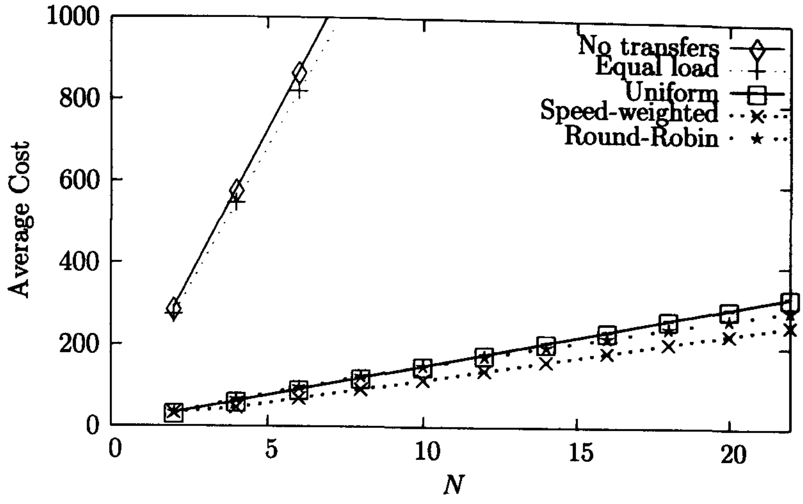

Figure 6.6: Policy comparison for increasing number of queues
$\lambda_{2i} = 1.875$, $\lambda_{2i+1} = 0.125$, $\mu_{2i} = 5$, $\mu_{2i+1} = 1$, $\xi_i = \eta_i = 0.1$

The comparisons illustrated in Figure 6.6 show similar trends to those in Figure 6.5. The differences between the three best policies become more pronounced for higher values of N. The *Speed-weighted* policy outperforms *Round-Robin*, which in turn outperforms the *Uniform* policy. This is not too surprising.

It should be noted that the *Fastest Other* policy has not been plotted on this graph. This is because, though the policy performs well for $N \leq 6$, for larger numbers of servers, the system is saturated, as all jobs which time-out are transferred to either queue 1 or queue 2, and they remain within this pair, transferring from 1 to 2 and back again until service commences, whereas all other policies allow jobs to transfer to all servers.

6.6 Conclusions

In this chapter, the approximate iterative approach was used to determine behaviour of the system as the transfer rate from a server is varied, and hence the optimal value for a given set of parameters. This was then used to compare a number of policies, one of which tends to outperform all others under consideration.

The approximation used in this chapter, of not timing out when operative and quickly clearing the queue when inoperative, assumes that the operative and inoperative periods are large and performs very badly when this assumption does not hold.

When the overall system load is high the difference between the optimal transfer policy and the simple heuristic increases. In general, the approximation performs well and is a computationally inexpensive heuristic.

Chapter 7

Conclusions

7.1 Contributions

This thesis has been concerned with methods for balancing the offered load between M/M/1 servers with negative exponentially distributed breakdowns and subsequent repairs. Two approaches are considered to achieve a balanced offered load.

In Chapters 2 and 3 the problem under consideration was that of dynamic routing where incoming jobs are routed to one of the servers in the system upon arrival, remaining with this server until service is complete. This approach is sensible for situations where server states are known centrally, and the cost of centralised transfers is low. This model lends itself to exact solution and in Chapter 2, three approaches to compute the optimal routing policy were presented and evaluated. However, this computation suffered from complexity issues due to the size of the state space, and the necessary number of iterations to converge to the optimal policy. So, for practical purposes, a number of easily computed heuristic policies were described in Chapter 3. When compared to the optimal policy two of these were found to compare favourably, as suggested by various simulation results.

In Chapter 4, a preliminary system for balancing the offered load using dynamic job transfers was considered. A system of two servers allowing job transfers in only one direction was considered, with servers always available. The benefit of this system over a more general case is that the system is exactly solvable. The optimal transfer rate for a range of system parameters was computed by solving the balance equations for the system as a whole, using the technique of spectral expansion. The behaviour of the exact solution was also compared against an approximate technique, and shown to be in good agreement.

This system was then extended in Chapter 5 to a system with general transfers between N always available servers. As the system is now no longer exactly solvable, the approximate technique was used to determine the distribution of jobs in the system and hence the average number of jobs in the system and the average transfer rate in the system. A number of easily computed heuristic policies for determining the transfer destination were evaluated and compared with an 'optimal' transfer policy, obtained by a brute force search through possible policies, and several were found to perform well.

Finally, in Chapter 6, the problem was extended further to allow the servers to go through periods of availability and unavailability. The approximate technique was used to compare a number of different heuristics with the same type of 'optimal' transfer policy as in the previous chapter. As before, a heuristic policy for the transfer destination was presented which performed well in comparison to the optimal and which was significantly better over a range of parameters than the other heuristics under consideration.

7.2 Application

In the introduction, the applicability of this research was targeted on distributed computing and in particular the Computing Grid. This is a diverse and loosely connected research area, and so a discussion of the applicability of this research will consider one field, that of mobile ad-hoc networks.

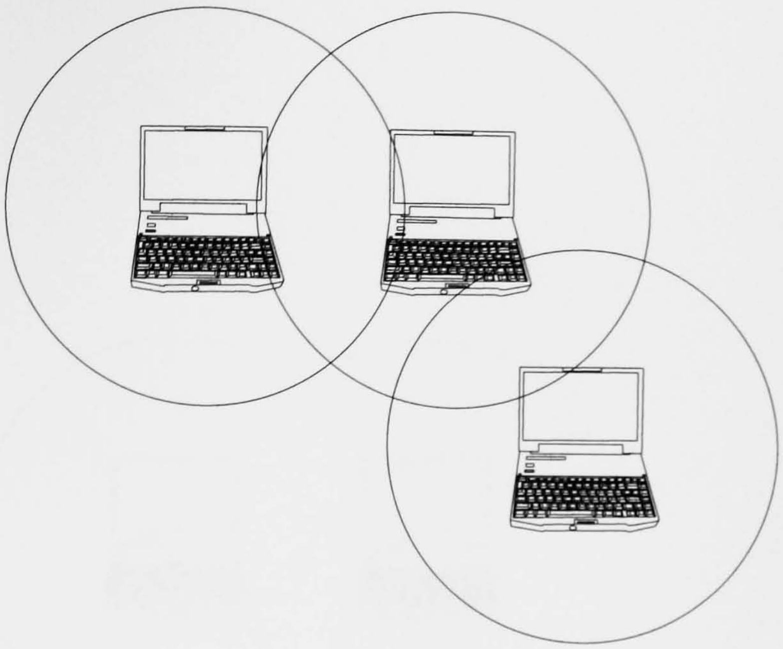

Figure 7.1: A mobile ad-hoc network, with all routers connected

A mobile ad-hoc network is a wireless network of mobile routers connected by wireless links. As the routers are free to move and connect to a varying number of other routers based on some criterion, usually distance, the network topology can change rapidly and unpredictably (see Figure 7.1).

As the presence of a connection between a pair of routers is governed by the distance between them, it is possible that a router may become entirely separated from the rest of the network, or that the network itself may fracture into sub-networks which are disconnected from one another (see Figure 7.2.

Considering the situation where a stream of data is to be sent from an originating server through the ad-hoc network to a destination server, each packet can be treated as a job. If the order of packets may be received in any order, they can be routed

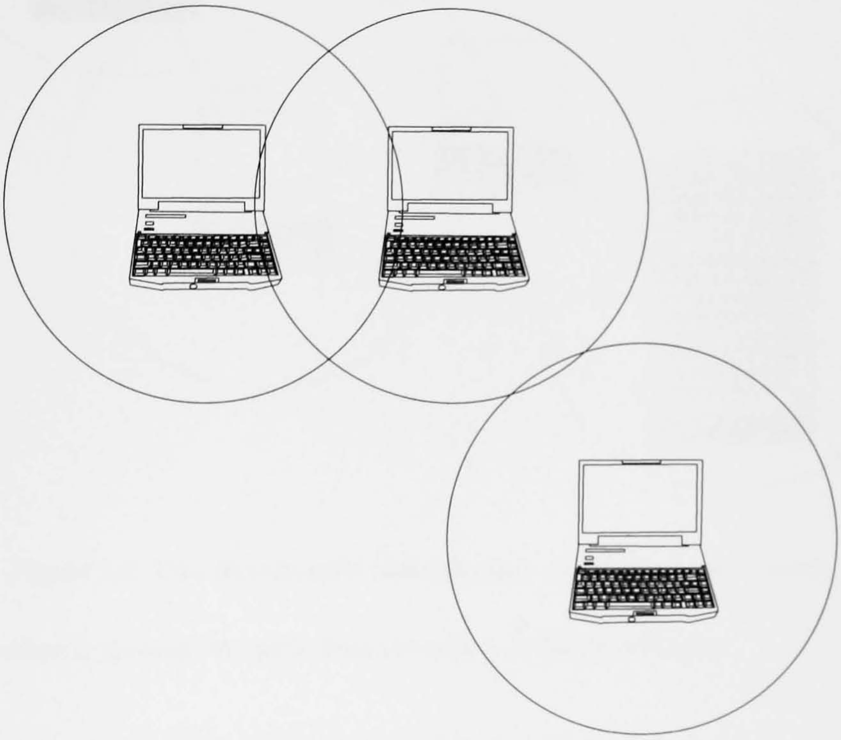

Figure 7.2: A mobile ad-hoc network, with one router out of contact

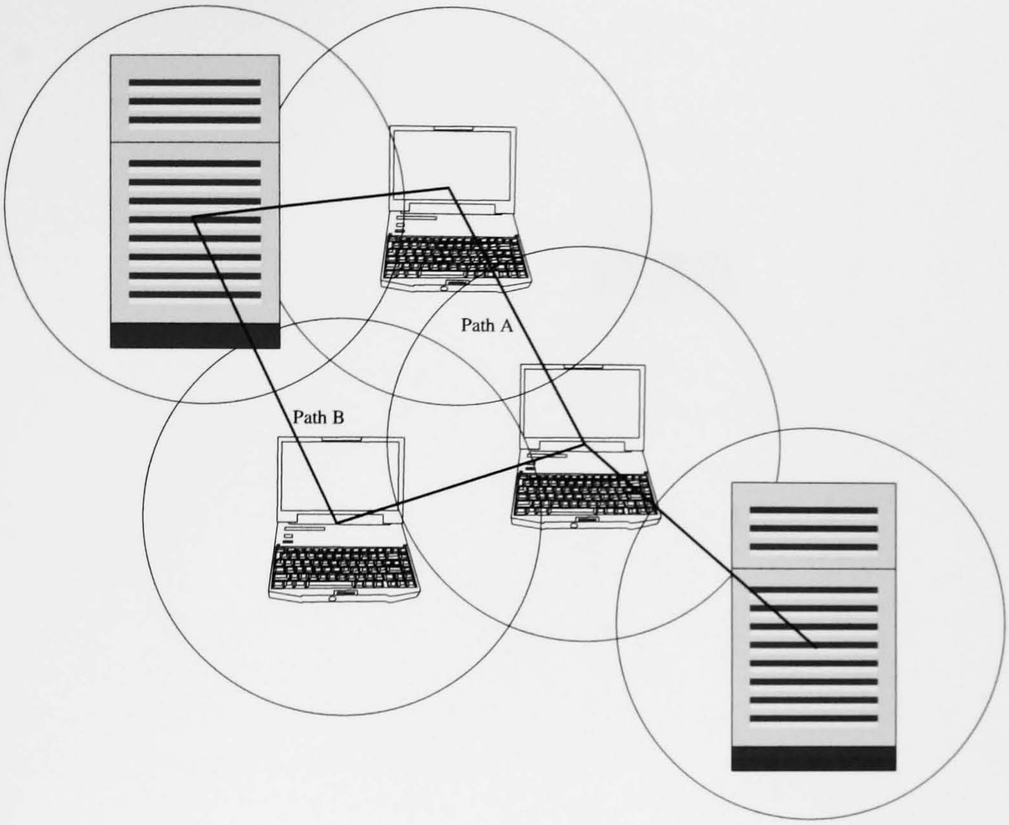

Figure 7.3: Two servers, with paths through a mobile ad-hoc network

independently through any path from the source to the destination.

From the point of view of the transmission end-points, the job can be considered complete when it has successfully reached the destination. Therefore, it is reasonable to treat a path which the packet takes through the ad-hoc network to be a server (see Figure 7.3).

As a server in this application consists of a number of routers and the transmission between them, the transmission time from source to destination along a particular path is equivalent to the average service time of a server in this thesis.

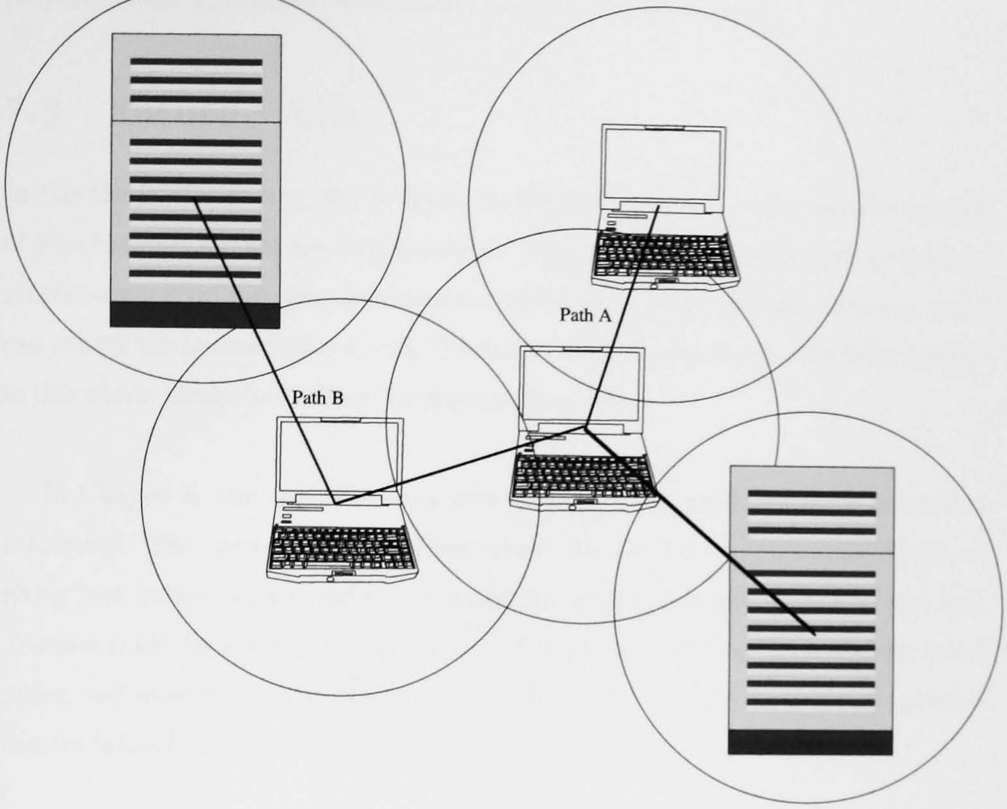

Figure 7.4: Two servers, with one interrupted path and one available path through a mobile ad-hoc network

When a router within the ad-hoc network changes location enough to affect the network topology, it can cause the selected path between origin and destination to no longer be valid, which maps in this body of work to a server becoming unavailable.

If a path becomes unavailable, the packet stream must either take an alternate route to the destination, which is equivalent to a transfer between servers, or wait for the path to be restored. In the case where an alternate route is taken, packets which already started transmission experience a transfer delay by undergoing a transfer route onto the new route, which may be of zero length and hence delay. Upon completion of this transfer delay the packets will join the new route. This transfer delay is a property of the application which is not modelled in this research.

7.3 Future Work

In this thesis, the routing of jobs from the dispatcher to the servers and the transfer of jobs between servers are instantaneous. This is not very realistic and as the geographic separation between the source and destination of the transfer increases, so we can expect the transfer time to rise. Therefore, a good extension to the work covered in this thesis would be to allow for transmission delays.

In Chapter 6, the case of servers with long operative and inoperative periods is considered. This generates the problem where the specific response time of jobs arriving just before failure, and the utilisation of servers just after repair is very bad. Transfer could potentially reduce this by transferring jobs away from recently failed nodes, and towards recently repaired ones. This would be an interesting and probably fruitful further investigation.

Additionally in this thesis, two approaches to load balancing have been considered and solved separately. There is no reason why a system could not be considered where small groups of servers, presumably in close geographical proximity, could be supplied with jobs from a local dispatcher, and jobs which reach their deadline would then be transferred between groups of servers.

In addition, every model in this thesis has assumed that the number of servers in the system is a constant. This does not reflect an observed feature of computational resources allocation that, as time goes by, the degree of interconnection between previously separate systems increases. Therefore, it may be of interest to consider a load balancing system where the total number of servers in the system increases, and the total incoming load also increases with time.

In this thesis it has been assumed that the information which decisions are based on is up to date. However, as geographical separation increases, this assumption will become increasingly invalid. Thomas, Bradley and Knottenbelt [21] looked at this problem in a static allocation model, and showed that a small delay in failure information propagation can have a significant negative impact.

Both of these models could be analysed using the techniques developed and discussed in this thesis. Especially in the former case, the model description and hence the computation requirements would be higher than in the models considered here.

Bibliography

[1] Eithan Altman and Uri Yechiali. Analysis of customers' impatience in queues with server vacations. *Queueing Systems*, 52:261–279, 2006.

[2] C.J. Ancker and A. Gafarian. Queueing with impatient customers who leave at random. *Journal of Industrial Engineering*, 13:84–90, 1962.

[3] Jongho Bae, Sunggon Kim, and Eui Yong Lee. The virtual waiting time of the M/G/1 queue with impatient customers. *Queueing Systems*, 38:485–494, 2001.

[4] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, 1995.

[5] Nam Kyoo Boots and Henk Tijms. A multiserver queueing system with impatient customers. *Manage. Sci.*, 45(3):444–448, 1999.

[6] Bong Dae Choi, Bara Kim, and Dongbi Zhu. MAP/M/c queue with constant impatient time. *Mathematics of Operations Research*, 29(2):309–325, May 2004.

[7] Amy Csizmar Dalal and Scott Jordan. Optimal scheduling in a queue with differentiated impatient users. *Performance Evaluation*, 59:73–84, 2005.

[8] Edmundo de Souza e Silva and H. Richard Gail. *Performability Modelling: Techniques and Tools*, chapter "The Uniformization Method in Performability Analysis", pages 31–41. Wiley, John & Sons, Incorporated, 2001.

[9] R.D. Foley and D.R. McDonald. Join the shortest queue: stability and exact asymptotics. *Ann. Appl. Probab.*, 11:569–607, 2001.

[10] K. D. Glazebrook and C. Kirkbride. Dynamic routing to heterogeneous collections of unreliable servers. *Queueing Systems*, 55(1):9–25, 2007.

[11] William J. Gray, P. Patrick Wang, and Meckinley Scott. A queueing model with multiple types of server breakdowns. *Quality Technology and Quantitative Management*, 1(2):245–255, 2004.

[12] Qi-Ming He and Marcel F. Neuts. Two M/M/1 queues with transfers of customers. *Queueing Systems*, 42:377–400, 2002.

[13] Liqiang Liu and Vidyadhar G. Kulkarni. Explicit solutions for the steady state distributions in M/PH/1 queues with workload dependent balking. *Queueing Systems*, 52:251–260, 2006.

[14] I. Mitrani. *The Spectral Expansion Solution Method for Markov Processes on Lattice Strips (in Advances in Queueing)*, chapter 13. CRC Press, 1995.

[15] I. Mitrani. Spectral expansion solutions for markov-modulated queues. *Springer LNCS 2459*, pages 17–35, 2002.

[16] I Mitrani and P.E. Wright. Routing in the presence of breakdowns. *Performance Evaluation*, 20:151–164, 1994.

[17] Ali Movaghar. On queueing with customer impatience until the beginning of service. *Queueing Systems*, 29:337–350, 1998.

[18] A. D. Polyanin and V. F. Zaitsev. *Handbook of Exact Solutions for Ordinary Differential Equations*. Chapman and Hall, 2003.

[19] S. M. Ross. *Introduction to Stochastic Dynamic Programming*. Academic Press, 1 edition, 1986.

[20] Nahum Shimkin and Avishai Mandelbaum. Rational abandonment from tele-queues: Nonlinear waiting costs with heterogeneous preferences. *Queueing Systems*, 47:117–146, 2004.

[21] N. Thomas, J. T. Bradley, and W. J. Knottenbelt. Stochastic analysis of scheduling strategies in a grid-based resource model. *IEE Software Engineering*, 151(5):232–239, 2004.

[22] N. Thomas and I. Mitrani. Routing among different nodes where servers break down without losing jobs. *IEEE International Computer Performance and Dependability Symposium (IPDS'95)*, page 0246, 1995.

[23] Kuo-Hsiung Wang, Tsung-Yin Wang, and Wen-Lea Pearn. Maximum entropy analysis to the N policy M/G/1 queueing system with server breakdowns and general startup times. *Applied Mathematics and Computation*, 165(1):45–61, 2004.

[24] Amy Ward and Peter Glynn. A diffusion approximation for a GI/GI/1 queue with balking or reneging. *Queueing Systems*, 50(4):371–400, August 2005.

[25] P. Whittle. *Optimization over Time: Dynamic Programming and Stochastic Control*, volume 1. John Wiley & Sons, 1982.

[26] P. Whittle. Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability*, A25:287–298, 1988.

[27] S.H. Xu and Y.Q. Zhao. Dynamic routing and jockeying controls in a two-station queueing system. *Adv. in Appl. Probab.*, 28:1201–1226, 1996.

[28] Sergey Zeltyn and Avishai Mandelbaum. Call centers with impatient customers: Many-server asymptotics of the M/M/n + G queue. *Queueing Systems*, 51(3-4):361–402, December 2005.

[29] Y. Zhao and W.K. Grassmann. Queueing analysis of a jockeying model. *Operations Research*, 43:520–529, 1995.