DYNAMIC ROUTING BETWEEN TWO QUEUES WITH UNRELIABLE SERVERS

Simon Martin and Isi Mitrani
School of Computing Science, University of Newcastle, NE1 7RU, UK
[s.p.martin,isi.mitrani]@ncl.ac.uk

Abstract:
We examine the problem of how best to route jobs among two queues whose servers are subject to breakdowns and repairs. The optimal routing policy is computed by modelling the system as a discrete-time, finite-state Markov decision process and solving the resulting dynamic programming equations either iteratively or by applying the 'policy improvement' algorithm. In a series of numerical experiments, the performance of various heuristic policies is compared with that of the optimal policy. A particular heuristic is shown to be close to optimal over a wide range of parameters.

Keywords: *Unreliable servers, Optimal routing, Grid computing, Dynamic programming, Heuristic policies.*

1 Introduction

Recent developments in distributed processing, and in particular the emergence of the *Computing Grid*, enable users to submit jobs without specifying where they will be executed. A grid-based system might contain a number of (geographically distributed) servers with different characteristics. A dispatcher handles the submitted jobs and decides where to send each one, taking into account information about the current system state. That information typically includes queue sizes, and also the availability or otherwise of the servers. In such an environment, one cannot assume that all servers are available at all times to run submitted jobs. A more realistic assumption is that each server goes through alternating periods of being available and unavailable, or 'operative' and 'inoperative' (in practice, an inoperative period is usually due to the server being occupied with other tasks of higher priority; however, as far as the user jobs are concerned, it 'breaks down' and is later 'repaired').

Thus, it is important to design dispatching (routing) policies that are (a) dynamic, i.e. react to changes in system state, and (b) efficient, i.e. make optimal or nearly optimal routing decisions with respect to some Quality of Service (QoS) criterion. That is the aim of this paper.

The above dynamic optimization problem has not, as far as we know, been tackled before. Interest in single queues subject to breakdowns and repairs goes back more than forty years (e.g., see [Gaver, 1962]). [Fiems et al, 2004] evaluate a single server experiencing interruptions in service, for the cases of the interrupted job being resumed, partially resumed or restarted.

However, there are very few studies involving more than one queue. [Mitrani and Wright, 1994] examined a model with N queues where a server breakdown results in the loss of all jobs in the associated queue. [Thomas and Mitrani, 1995] considered a similar system without job losses. In both those systems the routing policies are semi-static: routing decisions depend on the current operative state of the servers, but not on the sizes of the queues, whereas in the current work, the routing decisions can also depend on the sizes of the queues.

[He and Neuts, 2002] deal with a model of two servers with independent queues, where jobs are transferred from the server with the longer queue when the difference in queue lengths is greater than some threshold.

Here we study a Markovian model where in-

coming jobs can be directed to one of two queues, with different, intermittently available servers. Once dispatched, jobs remain in the appropriate queue until eventually completed. A 'holding cost' (possibly different for the two queues) is incurred by each job per unit time spent in the system. The objective is to find a routing policy that minimizes the total (possibly discounted) cost incurred over a finite or infinite planning horizon. That means, in essence, minimizing a weighted average number of jobs present in the system.

The solution approach is to (i) 'uniformize' the continuous time Markov decision process in order to formulate the problem in discrete time, (ii) truncate the state space in order to make it finite, and (iii) solve the resulting set of dynamic programming equations either by iteration or by employing a 'policy improvement' algorithm.

The optimal routing policy is difficult to implement because it does not lend itself to explicit characterization. The routing decisions are sometimes counter-intuitive: e.g., it may be preferable to send jobs to a longer queue, or to one where the server is currently inoperative, in the expectation that faster or more reliable service will be obtained later. However, a numerical computation of the optimal policy provides a standard of comparison by means of which other, easily implementable heuristic policies can be judged. Several such heuristics are evaluated. One, in particular, is shown to be close to optimal over a wide range of parameters.

The mathematical model is described in section 2. Its solution and the computation of the optimal policy are discussed in section 3. The results of a number of numerical experiments, including comparisons between the optimal and heuristic policies, are presented in section 4.

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 two 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 $1/\mu_i$ ($i = 1, 2$). 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 1.

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 two 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 quadruple of integers: $S = (j_1, j_2, b_1, b_2)$, where j_i is the current number of jobs in queue i, and b_i is the current availability of server i ($i = 1, 2$); 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, a, is defined by specifying, for every state S, the action, a_S, taken when a job arrives and finds that state: $a_S = i$ 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, $r_a(S, S')$, from state S to state S' under policy a, is given by

$$r_a(S,S') = \begin{cases} \lambda & \text{if } a_S = i \text{ and } S' = S + e_i; \\ \mu_i b_i & \text{if } j_i > 0 \text{ and } S' = S - e_i; \\ \xi_i & \text{if } b_i = 1 \text{ and } S' = S - e_{i+2}; \\ \eta_i & \text{if } b_i = 0 \text{ and } S' = S + e_{i+2}; \\ 0 & \text{otherwise} \end{cases} , \quad (1)$$

where $i = 1, 2$; e_k is the 4-dimensional vector whose kth element is 1 and the other three are 0.

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

$$r(S) = \lambda + b_1[\mu_1\delta(j_1 > 0) + \xi_1] + b_2[\mu_2\delta(j_2 > 0) + \xi_2] + (1 - b_1)\eta_1 + (1 - b_2)\eta_2 , \quad (2)$$

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 $r(S)$

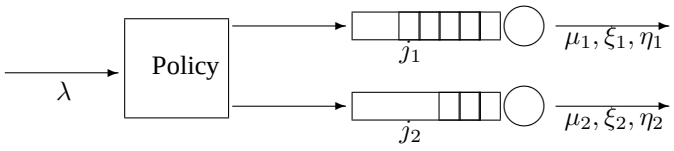

Figure 1. Routing among two heterogeneous queues

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 the other remains empty, then the system should be stable if the arrival rate is lower than the total average available service capacity:

$$\lambda < \frac{\eta_1}{\xi_1 + \eta_1}\mu_1 + \frac{\eta_2}{\xi_2 + \eta_2}\mu_2 \ . \tag{3}$$

3 Computation of the optimal policy

For the purposes of optimization, it is convenient to apply the technique of uniformization to the Markov process (e.g., see [de Souza e Silva and Gail, 2001]). 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

$$\Lambda = \lambda + \max(\mu_1 + \xi_1, \eta_1) + \max(\mu_2 + \xi_2, \eta_2) \ . \tag{4}$$

Without loss of generality, the unit of time can be scaled so that the right-hand side of (4) 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 a, $q_a(S, S')$, are equal to

$$q_a(S, S') = \begin{cases} r_a(S, S')/\Lambda & \text{if } S' \neq S \\ 1 - r(S)/\Lambda & \text{if } S' = S \end{cases} \ , \tag{5}$$

with $r_a(S, S')$ and $r(S)$ given by (1) and (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

$$v_0(S) = c_1 j_1 + c_2 j_2 \ . \tag{6}$$

If the state is S and an arrival occurs, then in addition to $v_0(S)$, a holding cost equal to c_1 if $a_S = 1$, and to c_2 if $a_S = 2$, 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. These costs satisfy a set of dynamic programming equations (for the general theory, see [Ross, 1986, Whittle, 1982]).

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

$$V_n(S) = v_0(S) + \lambda V_{n-1}^a(S) + \sum_{S'} q(S, S') V_{n-1}(S') \ , \tag{7}$$

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 $V_{n-1}^a(S)$. The sum in the third term extends over the transitions $S \rightarrow S'$ which do not involve an arrival: the next state is S' with probability $q(S, S')$; 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 ($i = 1, 2$). The state then immediately jumps to $S + e_i$. The cost $V_n^a(S)$ 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:

$$V_n^a(S) = v_0(S) + \min_{i=1,2} \left[c_i + \lambda V_{n-1}^a(S+e_i) + \sum_{S'} q(S+e_i, S') V_{n-1}(S') \right] . \quad (8)$$

Again, the sum in the right-hand side extends over the transitions $S + e_i \rightarrow S'$ 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 $V_0^a(S) = v_0(S) + \min(c_1, c_2)$. In practice, the state space must be made finite by bounding the queue sizes: $j_1 \leq J_1$ and $j_2 \leq J_2$ for some J_1 and J_2. The consequences of such a truncation are that if $j_1 < J_1$ and $j_2 = J_2$, then new arrivals must be routed to queue 1; if $j_1 = J_1$ and $j_2 < J_2$, new arrivals must be routed to queue 2; if $j_1 = J_1$ and $j_2 = J_2$, new arrivals are lost.

The complexity of the iterative solution is of the order $O(J_1 J_2 n)$, since the size of the state space is $4J_1J_2$ and there are n steps (the summations in (7) and (8) 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 (8) 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 $\alpha \rightarrow 1$, the weight of a future step, no matter how distant, approaches that of the current one.

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

$$V(S) = v_0(S) + \alpha\lambda V^a(S) + \alpha \sum_{S'} q(S, S') V(S') , \quad (9)$$

$$V^a(S) = v_0(S) + \min_{i=1,2} \left[c_i + \alpha\lambda V^a(S+e_i) + \alpha \sum_{S'} q(S+e_i, S') V(S') \right] , \quad (10)$$

with the same restrictions on S' as for (7) and (8), 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 (10).

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 $\alpha \rightarrow 1$. Three methods for computing the optimal policy numerically are described below. In all cases, the state space is truncated by introducing the bounds $j_1 \leq J_1$ and $j_2 \leq J_2$, and imposing the appropriate policy restrictions on the boundaries $j_1 = J_1$ and $j_2 = J_2$ (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 $n \rightarrow \infty$. The algorithm works as follows:

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

2. At iteration n, compute $V_n(S)$ and $V_n^a(S)$ according to (7) and (8) respectively, using $V_{n-1}(S)$ and $V_{n-1}^a(S)$ from iteration $n-1$. Terminate when

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

   for some small ϵ.

3. Return the policy specified by the values of i which achieve the minima in the right-hand side of (8) on the last iteration.

The complexity of the Cost Iteration algorithm is of the order $O(J_1 J_2 n)$, where n is the number of itera-

tion 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 $O(J_1 J_2 n)$, 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 $f(S) = 1$ if $j_1 < j_2$, 2 otherwise (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 (9) and (10) is that in the right-hand side of (10) there is no $\min_i$; the routing action $f(S)$ is used. This new version of (9) and (10) 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 (10). 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 $O(J_1 J_2 mn)$, 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 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 1. For each state where the queue sizes are in the range 0-20, 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 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.

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

j_1 \ j_2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	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	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
13	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
14	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
15	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
17	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
19	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
20	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1

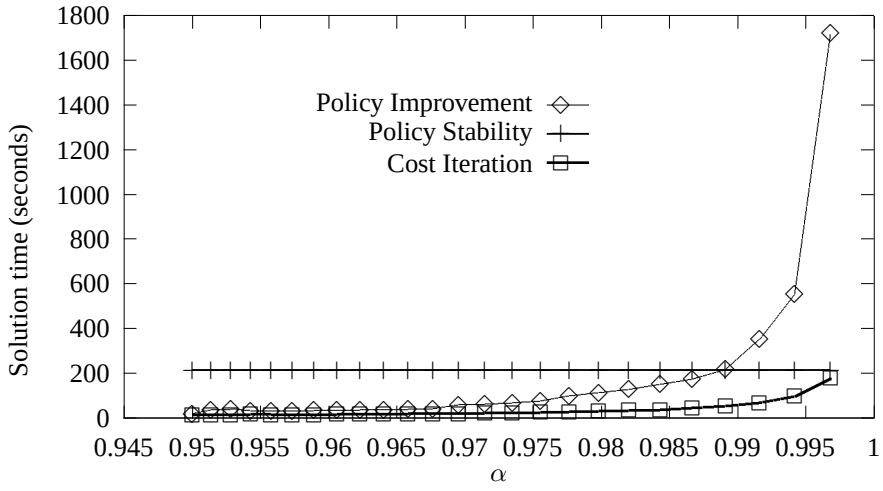

Figure 2. Run times of different solution methods

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 parameters 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 section.

4 Experimental results

Athough the terms 'breakdown rate' and 'repair rate' are used for ξ and η in the discussion which follows, it should be remembered that in practice those are rates at which servers become 'unavailable' and 'unavailable', respectively. The actual reasons for unavailability are not important here.

A number of numerical experiments were carried out, with the following aims:

(a) Compare the different methods for determining the optimal policy, with respect to both efficiency (measured by the run time) and behaviour as $\alpha \to 1$.

(b) Compare the performance of several heuristic routing policies with that of the optimal policy, for different parameter values.

In all cases, the state space was truncated at $J_1 = J_2 = 100$ (leading to a state space of size 40000). The arrival, service, breakdown and repair parameters were chosen so that, without the truncation, the probabilities of the queues reaching or exceeding those bounds would be small.

The run times of the Cost Iteration, Policy Stability and Policy Improvement solution algorithms, for different values of the discount factor α, are illustrated in figure 2.

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. 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, a, we use a single average cost metric, C_a, which is computed as follows. First, find the steady-state distribution, $\pi_a(S)$, of the system state under policy a. This is obtained by solving numerically the balance equations,

$$\pi_a(S) = \sum_{S'} \pi_a(S') q_a(S', S) \ , \qquad (11)$$

(where the one-step transition probabilities $q_a(S', S)$ are given by (5)), together with the normalizing equation,

$$\sum_S \pi_a(S) = 1 \ . \qquad (12)$$

The state space is truncated as before.

The average cost incurred under policy a per unit time, C_a, is then given by

$$C_a = \sum_S \pi_a(S)(c_1 j_1 + c_2 j_2) = c_1 L_{a,1} + c_2 L_{a,2} \ , \qquad (13)$$

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

Figure 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.

The remarkable feature of the figure 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 $\alpha > 0.99$.

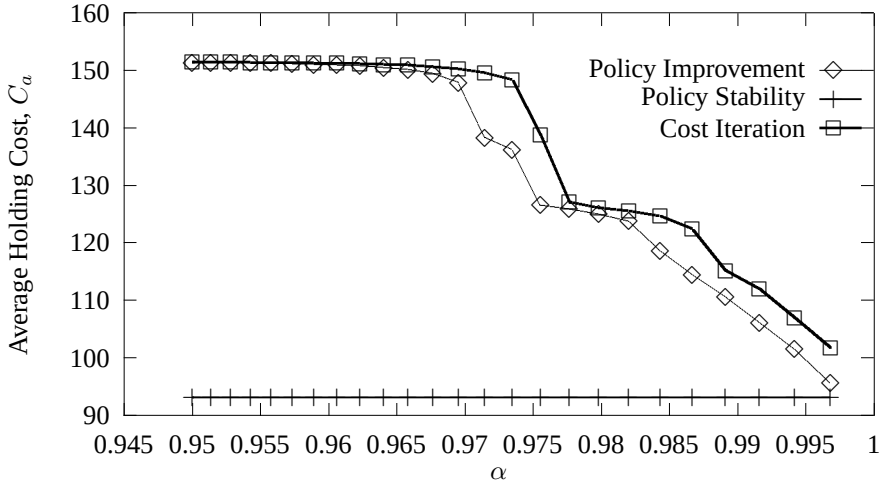

Figure 3. Effect of α on optimal policy

The remaining experiments compare the average cost of the optimal policy for $\alpha = 1$, obtained by the Policy Stability algorithm, with that of four simple heuristics. These are:

1. *Random* routing. Send new jobs to queue i with probability 0.5 ($i = 1, 2$), regardless of the system state.

2. *Selective* routing. If server 1 is operative and server 2 is not, send jobs to queue 1; if server 2 is operative and server 1 is not, send jobs to queue 2; otherwise send jobs to queue i with probability 0.5 ($i = 1, 2$). This is one of a class of selective policies introduced in [Thomas and Mitrani, 1995].

3. *Shortest Queue* routing. Send jobs to queue 1 if $j_1 < j_2$; otherwise to queue 2.

4. *Lowest Expected Cost* routing. If a job finds state S on arrival, evaluate the expected non-discounted cost, $d(S, i)$, it would incur if sent to queue i:

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

Send the job to queue 1 if $d(S, 1) < d(S, 2)$, otherwise to queue 2.

These are merely example heuristics, and any policy can in fact be used, provided that, for a given state S, it decides which server to route incoming jobs to.

The expression for $d(S, i)$ is explained by remarking that each service in queue i, of average length $1/\mu_i$, is interrupted ξ_i/μ_i times on the average, for an average interval $1/\eta_i$ per interrupt; in addition, if a job is sent to queue i when its server is inoperative, there is an average delay of $1/\eta_i$.

Figure 4 illustrates the comparisons 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 unit holding costs are equal: $c_1 = c_2 = 1$. The arrival rate λ is varied; the system is stable when $\lambda < 1.83$.

We observe that the two dynamic heuristics – Shortest Queue and Lowest Expected Cost – are almost optimal. The static (Random), and semi-static (Selective) policies are less efficient, particularly as the offered load increases. In this model, the Random policy becomes better than the Selective one at heavier loads.

The next experiment compares the performance of the optimal and heuristic policies when the asymmetry between the servers increases. The arrival rate is fixed at $\lambda = 4$, 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

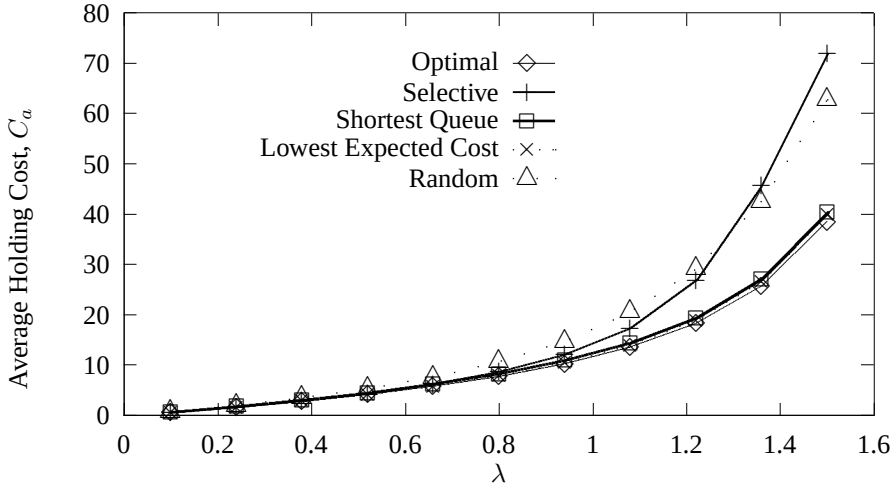

Figure 4. Performance of optimal and heuristic policies; different arrival rates

is the service rate at server 2, μ_2. Both unit holding costs are equal to 1. The results are shown in figure 5.

As expected, the average costs decrease when a service rate increases. The Lowest Expected Cost heuristic is again almost optimal, but now the Shortest Queue policy is noticeably poorer, while the Random and Selective policies are much worse.

Another way to increase 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 6. 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, $\xi_1 = 0.3$, $\eta_1 = 0.1$. The average operative and inoperative intervals of server 2 increase (ξ_2 and η_2 decrease), while their ratio remains fixed, $\xi_2 = 3\eta_2$. Again, $c_1 = c_2 = 1$.

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

Both the Lowest Expected Cost and the Shortest Queue policies perform well. The former starts off being considerably better, and ends up being very slightly worse, than the latter. The Random and Selective policies perform significantly worse over the entire range.

It should be pointed out that Figures 4, 5 and 6 are not entirely fair to the Random and Selective policies. Rather than sending jobs to the two queues with equal probabilities, one could choose the routing probabilities more intelligently (e.g., in proportion to their service rates if operative, or in proportion to their service and repair rates, if broken). Some limited experimentation has demonstrated that such choices do indeed improve the performance of the non-dynamic policies. However, even with those improvements, they are still significantly worse than the dynamic policies.

5 Conclusions

The numerical computation of the optimal routing policy can be a memory-consuming and time-consuming task. It is probably impractical to design dispatchers that either perform the optimization on-line, or use table look-up with pre-computed optimal policies. In practice, one would use some simple and easily implementable heuristic. However, being able to compute the optimal policy off-line is a useful means of evaluating the quality of different heuristics. The best candidates appear to be the Lowest Expected Cost and the Shortest Queue policies. We have shown that the first of these is nearly optimal over a wide range of parameter values. That policy

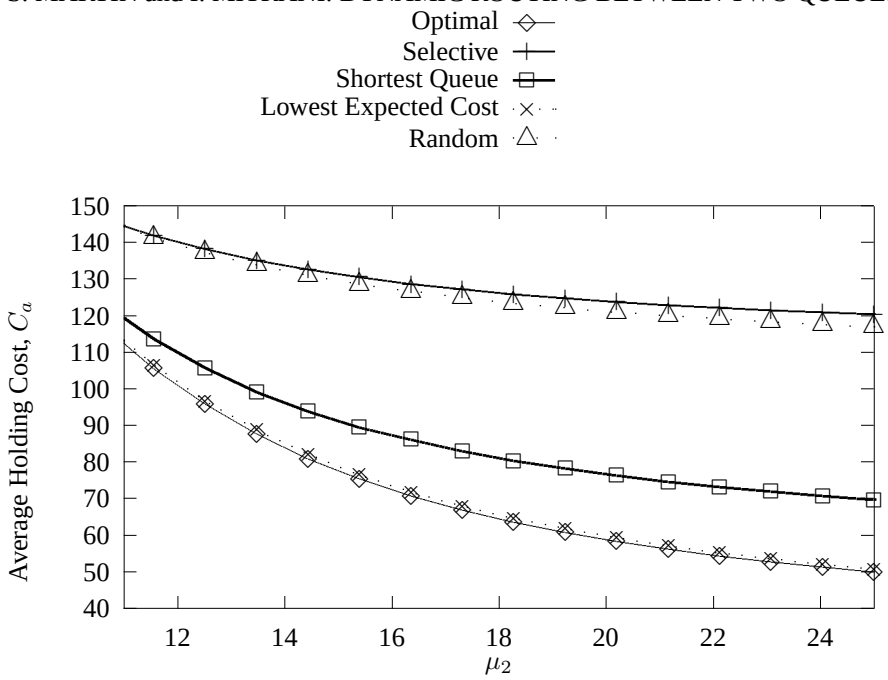

Figure 5. Performance comparison: changing service rate at server 2

is easily implementable, but requires knowledge of all parameters. The latter would have to be estimated by monitoring the system and collecting statistics about traffic, service times, operative and inoperative periods. On the other hand, the Shortest Queue policy tends to have a worse performance, but needs to know only the current queue sizes, not the values of parameters.

The obvious next stage in the investigation is to extend the methodology and results described here to systems with arbitrary number of queues. That will be the aim of future work.

References

[Fiems et al, 2004] D. Fiems, B. Steyaert and H. Bruneel, "Discrete-time queues with generally distributed service times and renewal-type server interruptions", Performance Evaluation, 55, 277-298, 2004

[Gaver, 1962] D.P. Gaver, "A Waiting Line with Interrupted Service, Including Priorities", Journal of the Royal Statistical Society, B, 24, 1, 73-90, 1962

[He and Neuts, 2002] Q.-M. He and M.F. Neuts, "Two M/M/1 Queues with Transfers of Customers", Queueing Systems, 42, 377-400, 2002

[Koole, 1997] G. Koole, "Assigning a single server to inhomogeneous queues with switching costs", Theoretical Computer Science, 182, 203-216, 1997

[Mitrani and Wright, 1994] I. Mitrani and P.E. Wright, "Routing in the presence of breakdowns", Performance Evaluation, 20, 151-164, 1994

[Palmer and Mitrani, 2004] J. Palmer and I. Mitrani, "Dynamic Server Allocation in Heterogeneous Clusters", 2004

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

[de Souza e Silva and Gail, 2001] E. de Souza e Silva and H.R. Gail, "The Uniformization Method in Performability Analysis", in *Performability Modelling* (eds B.R. Haverkort, R. Marie, G. Rubino and K. Trivedi), Wiley, 2001

[Thomas and Mitrani, 1995] N. Thomas and I. Mitrani, "Routing Among Different Nodes Where Servers Break Down Without Losing Jobs", Procs., IPDS'95, Erlangen, 1995

[Whittle, 1982] P. Whittle, *Optimization over Time*, Vol. 1, 1982

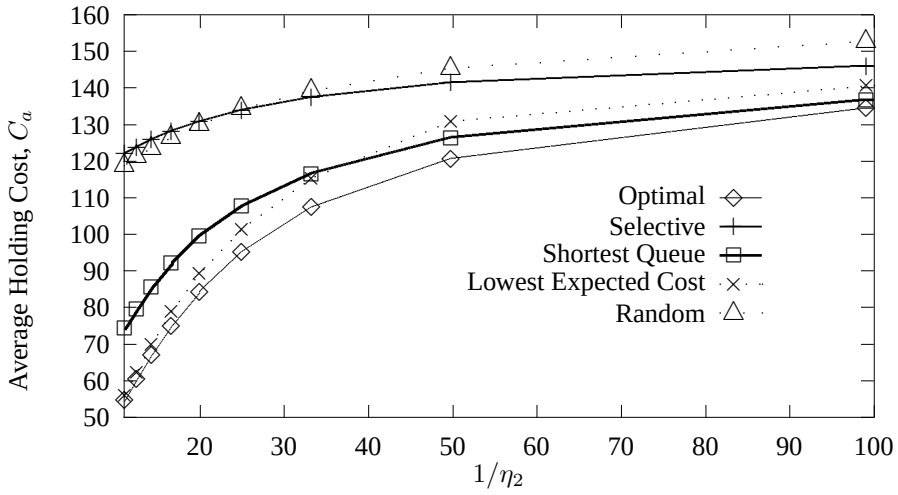

Figure 6. Increasing operative and inoperative intervals at server 2

Simon Martin is a PhD student in the School of Computing Science, University of Newcastle upon Tyne. He gained an MSci in Theoretical Physics from the University of Durham in 2001.

Isi Mitrani is Professor in the School of Computing Science, University of Newcastle upon Tyne, where he obtained his PhD in 1973. His research interests are in the areas of queueing theory, probabilistic modelling of computer and communication systems, and simulation. For further information, see www.cs.ncl.ac.uk/people/home.php?id=12