

Optimal Server Allocation in Reconfigurable Clusters with Multiple Job Types

J. Palmer I. Mitrani
jennie.palmer@ncl.ac.uk isi.mitrani@ncl.ac.uk

School of Computing Science, University of Newcastle,
NE1 7RU, UK

Abstract. We examine a system where the servers in a cluster may be switched dynamically and preemptively from one kind of work to another. The demand consists of M job types joining separate queues, with different arrival and service characteristics, and also different relative importance represented by appropriate holding costs. The switching of a server from queue i to queue j incurs a cost which may be monetary or may involve a period of unavailability. The optimal switching policy is obtained numerically by solving a dynamic programming equation. Two simple heuristic policies – one static and one dynamic – are evaluated by simulation and are compared to the optimal policy. The dynamic heuristic is shown to perform well over a range of parameters, including changes in demand.

Keywords: Optimal server allocation, Grid computing, Dynamic programming, Heuristic policies.

1 Introduction

This paper is motivated by recent developments in distributed processing, and in particular by the emerging concept of a *Computing Grid*. In a Grid environment, heterogeneous clusters of servers provide a variety of services to widely distributed user communities. Users submit jobs without necessarily knowing, or caring, where they will be executed. The system distributes those jobs among the servers, attempting to make the best possible use of the available resources and provide the best possible quality of service.

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

We consider a system consisting of a pool of N machines, split into M heterogeneous clusters of sizes K_1, K_2, \dots, K_M , where $\sum_{i=1}^M K_i = N$. Cluster i is dedicated to a queue of jobs of type i ($i = 1, \dots, M$). Job types may for example

include short web accesses or long database searches. Different types of job have different response time requirements (e.g., some may be less tolerant of delays than others). It is possible to reassign any server from one queue to another, but the process is generally not instantaneous and during it the server becomes unavailable. In those circumstances, a reconfiguration policy would specify, for any given parameter set (including costs), and current state, whether to switch a server or not.

There is an extensive literature on dynamic optimization (some good general texts are [1, 12, 14]), but the problem described here does not appear to have been studied before. There is a body of work on optimal allocation in the context of polling systems, where a server visits several queues in a fixed or variable order, with or without switching overheads (see [4, 5, 8–10]). Even in those cases of a single server, it has been observed by both Duenyas and Van Oyen [4, 5], and Koole [8, 9], that the presence of non-zero switching times makes the optimal policy very difficult to characterize explicitly. This necessitates the consideration of heuristic policies. The only general result available for multiprocessor systems applies when the switching times and costs are zero: then the $c\mu$ -rule is optimal, i.e. the best policy is to give absolute preemptive priority to the job type for which the product of holding cost and service rate is largest (Buyukkoc et al [3]).

A preliminary study of a model with just two job types was presented in [11]. A model similar to ours, also concerned with just two job types, was analyzed by Fayolle et al [7]. There the policy is fixed (servers are switched instantaneously, and only when idle), and the object is to evaluate the system performance. The solution is complex and rather difficult to implement.

Posed in its full generality, this is a complex problem which is most unlikely to yield an exact and explicit solution. Our approach is to formulate the problem as a Markov decision process, and to assume that there is a stationary optimal policy. This policy can be computed numerically by truncating the state space to make it finite. We then propose some heuristic policies which, while not optimal, perform well and are easily implementable. The quality of the heuristics, compared to the optimal policy, is evaluated by simulation.

The model assumptions are described in section 2. The dynamic programming formulation leading to the optimal policy is presented in section 3, while section 4 presents a number of numerical and simulation experiments, including comparisons between the optimal and heuristic policies. Section 5 summarizes the results.

2 The model

Our model may be described as follows. Jobs of type i arrive according to an independent Poisson process with rate λ_i , and join a separate unbounded queue ($i = 1, 2, \dots, M$). Their required service times are distributed exponentially with mean $1/\mu_i$. The cost of keeping a type i job in the system is c_i per unit time ($i =$

1, 2, ..., M). These ‘holding’ costs reflect the relative importance, or willingness to wait, of the M job types.

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

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

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

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

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

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

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

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

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

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

3 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 [13]). This entails the introduction of ‘fictitious’ transitions which do not change the system state, so that the average interval between consecutive transitions ceases to depend on the state, and then embedding a discrete-time Markov chain at transition instants. First, we find a constant, Λ , such that $r_d(S) \leq \Lambda$ for all S and d . A suitable value for Λ is

$$\Lambda = \sum_{i=1}^M \lambda_i + N\mu + N\zeta, \quad (1)$$

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

Next, construct a Markov chain whose one-step transition probabilities when action d is taken, $q_d(S, S')$, are given by

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

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

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

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

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

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

$$V_n(S) = \sum_{i=1}^M j_i c_i + \min_d \left[c(d) + \alpha \sum_{S'} q_d(S, S') V_{n-1}(S') \right]. \quad (2)$$

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

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

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

An argument similar to the one preceding (2) leads to the following equation for $V(S)$:

$$V(S) = \sum_{i=1}^M j_i c_i + \min_d \left[c(d) + \alpha \sum_{S'} q_d(S, S') V(S') \right]. \quad (3)$$

The optimal policy (i.e. the best action in any given state) is specified by the value of d that achieves the minimum in the right-hand side of (3).

Equation (3) can be solved by applying the ‘policy improvement’ algorithm (see Dreyfus and Law [6]).

4 Experimental results

We start with a simple system with $N = 2$ and $M = 2$, where switches cost money but do not take time. Although this case is not of great practical interest, it is included as an illustration. The system state is described by a triple, $S = (j_1, j_2, k_1)$. The number of servers allocated to type 2 is $k_2 = N - k_1$. The uniformization constant is now $A = \lambda_1 + \lambda_2 + 2(\max(\mu_1, \mu_2))$. If action $d \neq 0$ is taken in state S , the value of k_i changes immediately as a switch initiated from queue i to queue j . Then a new state is entered after an exponentially distributed interval with mean $1/A$.

In this example, the arrival and service parameters of the two job types are the same, but waiting times for type 2 are twice as expensive as those for type 1. The discount factor is $\alpha = 0.95$. The stationary optimal policy for states where $k_1 = k_2 = 1$ is shown in table 1. The truncation level used in the computation

was $J = 30$, but the table stops at $j_1 = j_2 = 10$; the actions do not change beyond that level. Actions d are numbered as follows:

- d=0, do nothing;
- d=1, switch a server from queue 1 to queue 2;
- d=2, switch a server from queue 2 to queue 1.

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

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

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

The optimal policy for $k_1 = 0$ is to take action $d = 2$ when $j_1 = 1$ and $j_2 = 0$, or when $j_1 > 1$ and $j_2 < 2$. When $k_1 = 2$, it is optimal to take action $d = 1$ when $j_1 \leq 1$ and $j_2 > 0$, or when $j_1 > 1$ and $j_2 > 1$.

From now on, we examine models where switching takes non-zero time. To keep the number of parameters low, the monetary costs of switching will be assumed negligible, $c_{1,2} = c_{2,1} = 0$. The uniformization constant is given by (1), and the unit of time is chosen so that $A = 1$.

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

Another approach is to pre-compute the optimal policy for a wide range of parameter values, and store a collection of tables such as table 1. Then, having monitored the system and estimated its parameters, the optimal policy could be obtained by a table look-up. This is feasible, but will consume a lot of storage.

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

4.1 Heuristic policies

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

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

if all k_i are non-zero. If any k_i is zero, replace k_i with 1 and the largest k_j ($i \neq j$) by $k_j - 1$. Repeat this process until all k_i are non-zero. Having made the allocation, leave it fixed as long as the offered loads and costs remain the same. This will be referred to as the ‘static’ policy. It certainly has the virtue of simplicity, and also provides a comparator by which the benefits of dynamic reconfiguration can be measured.

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

1. Calculate the following for each of the $M(M-1)/2$ possible switches from queue a to queue b ($a \neq b$ and $k_a > 0$):

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

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

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

This rule is based on approximating the effects of a switch. If j_b jobs of type b are present and k_b servers are available for them, then the average queue b increment during an interval of length x may be estimated as $x[\lambda_b - \mu_b \min(k_b, j_b)]$. Similarly for queue a , except that if a server is switched from queue a then the available servers for this queue drops to $k_a - 1$. Thus, a server is switched if that switch would help to balance the holding costs, after taking account of its effect on the M queues. The above policy will be referred to as the ‘heuristic’.

The optimal, static and heuristic policies are compared by simulation. In order to model changes in demand, the simulation includes a sequence of phases, with λ_i changing values from one phase to the next. There are two possible values for each λ_i : a high rate and a low rate. In any one phase, a particular λ_i is set at the high rate, while the remaining λ_i are at the low rate. This models demand peaking for a particular job type over a period of time. The performance measure in all cases is the total average holding cost, i.e. the simulation estimate of $E(\sum_{i=1}^M c_i j_i)$. The average phase duration is 100.

In all experiments, the parameters given below are renormalized to make the uniformization constant, Λ , equal to 1.

In figure 1, the average cost is plotted against total load. Here the number of servers is fixed at $N = 4$ and $M = 2$. The following parameters are used: $\mu_1 = \mu_2 = 1$, $\zeta_{1,2} = \zeta_{2,1} = 0.1$, $c_1 = 2$, $c_2 = 1$. The offered load increases, approaching saturation. The arrival rates are in the ratio 1:100 during phase 1 and 100:1 during phase 2, and are increased to produce the increase in total load.

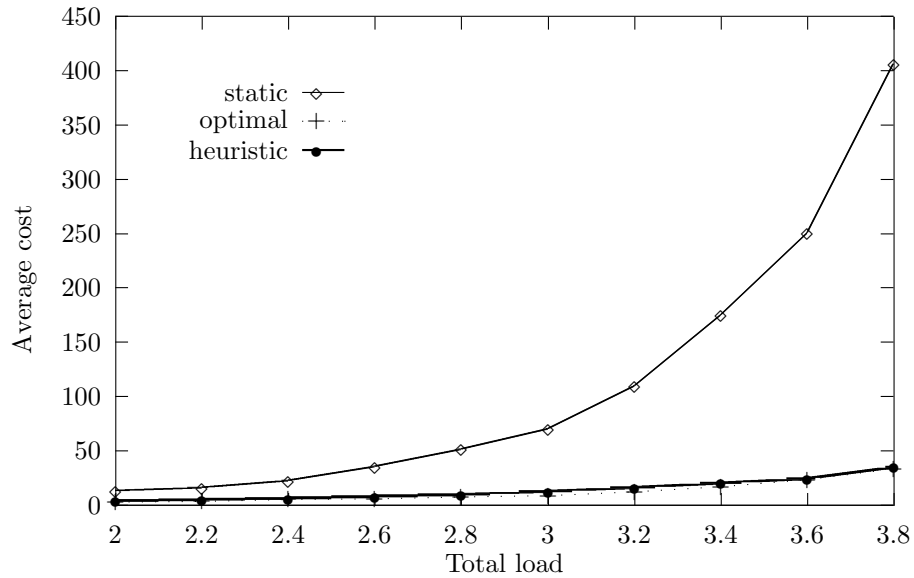


Fig. 1. Policy comparisons: $M = 2$ and increasing loads

This experiment shows emphatically that dynamic reconfiguration is advantageous. The cost of the static policy (which is not entirely static; it changes the allocation within each phase, as the arrival rates change) increases very quickly, while the heuristic, which is almost optimal, has much lower costs.

In figure 2, the average cost is plotted against the number of servers, N when $M = 3$. The following parameters are used: $1000\lambda_1 = \lambda_2 = \lambda_3$, $b_1 = 1000b_2 = 1000b_3$, $c_1 = 2$, $c_2 = 1$, $c_3 = 1$. Switching rates are equal and given by $\zeta_{i,j} = \mu_2/10 = \mu_3/10$. Arrival rates are increased with N so that the total offered load, $\rho_1 + \rho_2 + \rho_3$, is equal to $4N/5$ (i.e., the system is heavily loaded). There are no phase changes. This models a system where type 1 jobs are long and types 2 and 3 are much shorter. Requests of type 1 arrive at a much slower rate than for types 2 and 3, although the total load for each job type is the same.

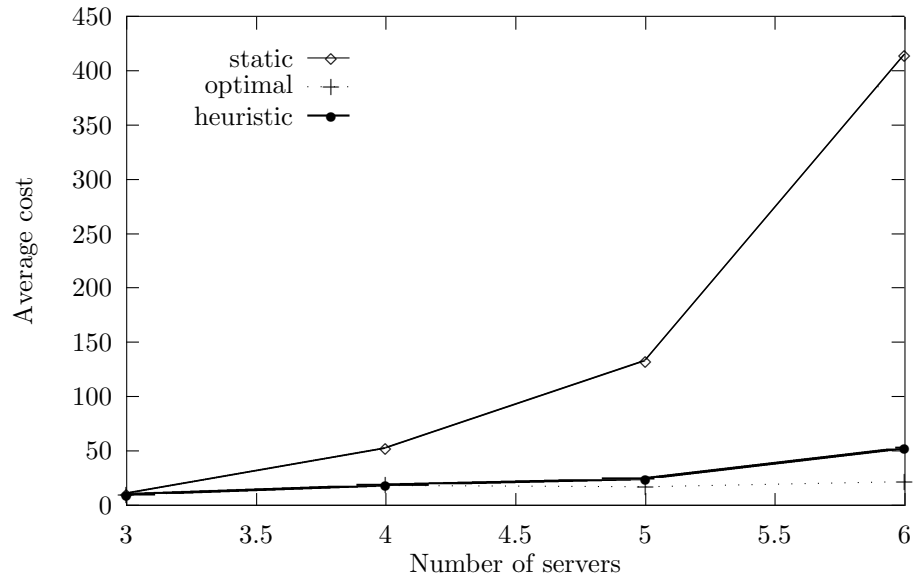


Fig. 2. Policy comparisons: $M = 3$ and increasing N

Another comparison when $M = 3$ is shown in figure 3. Here, the number of servers is fixed at $N = 4$ and the total load increases. Once again there are no phase changes, and all arrival rates are equal. The following parameters are used: $\mu_1 = \mu_2 = \mu_3 = 1$, $\zeta_{i,j} = 0.1$, $c_1 = 2$, $c_2 = 1$, $c_3 = 1$.

Figures 2 and 3 demonstrate clearly the benefit of dynamic reconfiguration of servers when the number of job types is increased to $M = 3$. The static policy performs poorly as the number of servers or the load is increased, while the heuristic policy performs almost as well as the optimal policy. In each case, choosing dynamic reconfiguration dramatically reduces the average holding costs.

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

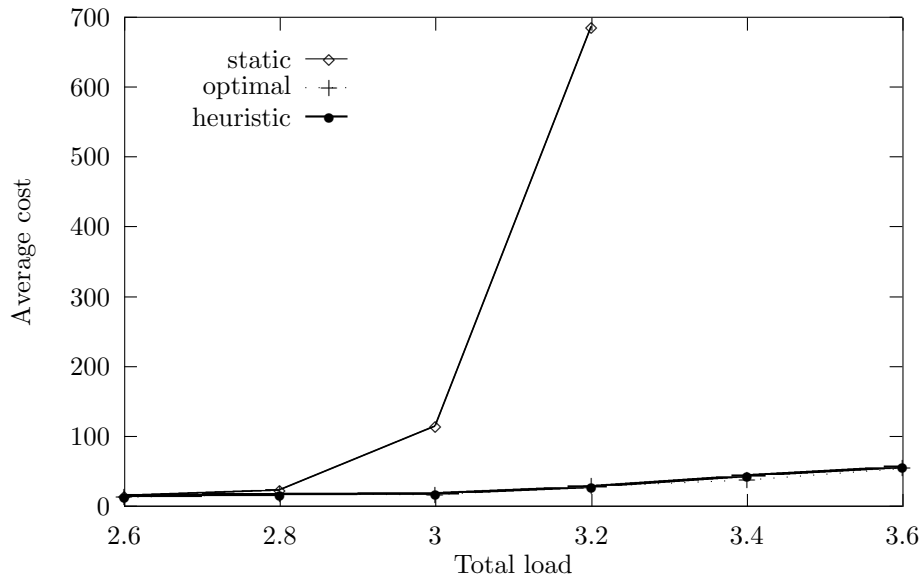


Fig. 3. Policy comparisons: $M = 3$ and increasing loads

Calculations of the table look-ups for the optimal policy have been executed on a Linux machine with an Intel Xeon 2.80GHz processor and 1GB RAM. To calculate the optimal policy for $N = 4$, $M = 3$ and a truncated queue size of $J = 30$ requires 223MB of available memory. As an illustration of the complexity of the calculations and the large size of the state space, if the number of servers is increased to $N = 6$, with $M = 3$ and $J = 30$, calculating the optimal policy now requires 1.564GB of available memory. Each table of decisions, when calculated using the Policy Improvement algorithm described in Section 3, took approximately 16 minutes. The Policy Improvement algorithm took 10 iterations to converge to the optimal policy, with the initial 'guess' of the policy set to the heuristic policy. Solving a large set of simultaneous equations using an iterative process is the most computationally expensive stage of the Policy Improvement algorithm. Initializing the cost matrix to the holding cost of the current state, this set of equations takes approximately 180 iterations to converge to within an accuracy of 0.01 in the first Policy Improvement iteration. This reduces upon each iteration.

5 Conclusions

A problem of interest in the area of distributed processing and dynamic Grid provision has been examined. The optimal reconfiguration policy can be computed and tabulated, subject to complexity constraints imposed by the size of the

state space and the ranges of parameter values. However, for practical purposes, an easily implementable heuristic policy is available. The encouraging results of figures 1-3 suggest that its performance compares quite favourably with that of the optimal policy.

Acknowledgement

This work was carried out as part of the collaborative project GridSHED (Grid Scheduling and Hosting Environment Development), funded by British Telecom and the North-East Regional e-Science centre.

References

1. J. Bather, *Decision Theory - An Introduction to Dynamic Programming and Sequential Decisions*, Wiley, 2000
2. D. Blackwell, "Discounted dynamic programming", *Annals of Mathematical Statistics*, 26, pp 226-235, 1965
3. C. Buyukkoc, P. Varaiya and J. Walrand, "The $c\mu$ -rule revisited", *Advances in Applied Probability*, 17, pp 237-238, 1985
4. I. Duenyas and M.P. Van Oyen, "Heuristic Scheduling of Parallel Heterogeneous Queues with Set-Ups", *Technical Report 92-60*, Department of Industrial and Operations Engineering, University of Michigan, 1992
5. I. Duenyas and M.P. Van Oyen, "Stochastic Scheduling of Parallel Queues with Set-Up Costs", *Queueing Systems Theory and Applications*, 19, pp 421-444, 1995
6. S.E. Dreyfus and A.M. Law, "The Art and Theory of Dynamic Programming", Academic Press, New York, 1977
7. G. Fayolle, P.J.B. King and I. Mitrani, "The Solution of Certain Two-Dimensional Markov Models", *Procs.*, 7th International Conference on Modelling and Performance Evaluation, Toronto, 1980
8. G. Koole, "Assigning a Single Server to Inhomogeneous Queues with Switching Costs", *Theoretical Computer Science*, 182, pp 203-216, 1997
9. G. Koole, "Structural Results for the Control of Queueing Systems using Event-Based Dynamic Programming", *Queueing Systems Theory and Applications*, 30, pp 323-339, 1998
10. Z. Liu, P. Nain, and D. Towsley, "On Optimal Polling Policies", *Queueing Systems Theory and Applications*, 11, pp 59-83, 1992
11. J. Palmer and I. Mitrani, "Dynamic Server Allocation in Heterogeneous Clusters", *Procs. of HETNETs '03 : First International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks*, pp. 12/1-12/10, UK, July 2003.
12. S. M. Ross, *Introduction to Stochastic Dynamic Programming*, Academic Press, 1983
13. 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
14. P. Whittle, *Optimisation over Time*, Vols 1 and 2, Wiley,