Dynamic Server Allocation in Heterogeneous Clusters

J. Palmer I. Mitrani

School of Computing Science, University of Newcastle, NE1 7RU, UK
jennie.palmer@ncl.ac.uk isi.mitrani@ncl.ac.uk

Abstract

We examine the optimization of 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 two job types joining separate queues, with different arrival and service characteristics, and also different relative importance represented by appropriate holding costs. The switching of a server from queue 1 to queue 2, or vice versa, incurs a cost which may be monetary or may involve a period of unavailability. The optimal switching policy in the case of bounded queues 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.

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 examine a simple, yet non-trivial special case, where the optimal dynamic reallocation policy can be computed numerically. We then propose some heuristic policies which, while

not optimal, perform reasonably well and are easily implementable. The quality of the heuristics, compared to the optimal policy, is evaluated by simulation.

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

Despite an extensive literature on dynamic optimization (some good general texts are [1, 11, 12]), 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, 9, 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 model similar to ours was analyzed by Fayolle et al [6]. There the policy is fixed (servers are switched instantaneously, and only when idle), and the object is to evaluate the system performance. The solution is complex and rather difficult to implement.

The 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 and outlines possible extensions.

2 The model

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

Any server currently allocated to queue 1 may be switched to queue 2. Such a switch costs $c_{1,2}$ and takes an interval of time distributed exponentially with mean $1/\xi$, during which the server cannot serve jobs. Similarly, a server allocated to queue 2 may be switched to queue 1, at the cost of $c_{2,1}$ and taking an interval of time distributed exponentially with mean $1/\eta$. It is assumed that switches are initiated at job arrival or departure instants. Indeed, it is at those

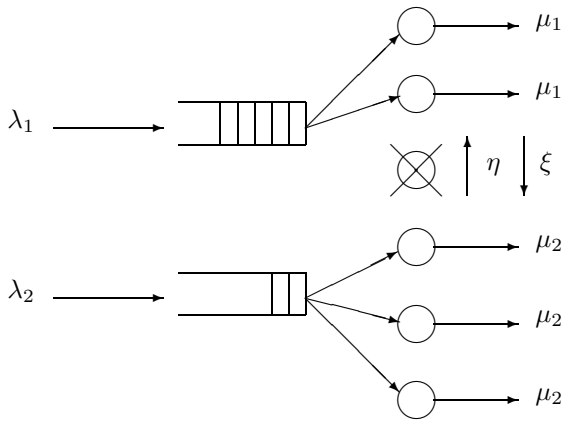

Figure 1: Two reconfigurable heterogeneous clusters

instants that switches may become advantageous, and if they do, they should be performed without delay. Also, it is assumed that the switching policy employed is memoryless, i.e., switching decisions may depend on the current state but not on past history.

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

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

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

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

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

The corresponding rates for $d = 1$ are obtained by replacing, in S', k_1 by

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

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

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

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 [7]). 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

$$\Lambda = \lambda_1 + \lambda_2 + N(\mu_1 + \mu_2 + \xi + \eta) \ . \tag{1}$$

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

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

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

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

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

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

Suppose that the action taken in state S is d. This incurs an immediate cost of $c(d)$, equal to $c_{1,2}$ if $d = 1$ and $c_{2,1}$ if $d = 2$. In addition, since the average interval between transitions is 1, each type 1 job in the system incurs a holding cost c_1 and each type 2 job in the system incurs a holding cost c_2. The next state will be S', with probability $q_d(S, S')$, and the minimal cost of the subsequent $n - 1$ steps will be $\alpha V_{n-1}(S')$. Hence, the quantities $V_n(S)$ satisfy the following recurrence relations:

$$V_n(S) = j_1 c_1 + j_2 c_2 + \min_d \left[c(d) + \alpha \sum_{S'} q_d(S, S') V_{n-1}(S') \right] \ . \tag{2}$$

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

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

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

$$V(S) = j_1c_1 + j_2c_2 + \min_d \left[c(d) + \alpha \sum_{S'} q_d(S, S')V(S') \right] . \tag{3}$$

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

4 Experimental results

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

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

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

		j_2										
		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	1	1	1	1	1	1	1	1
	1	0	0	0	0	0	1	1	1	1	1	1
	2	0	0	0	0	0	1	1	1	1	1	1
	3	0	0	0	0	0	1	1	1	1	1	1
	4	0	0	0	0	0	1	1	1	1	1	1
j_1	5	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, $N = 2$, $k_1 = 1$, $\lambda_1 = \lambda_2 = 0.086$, $\mu_1 = \mu_2 = 0.207$, $c_1 = 1$, $c_2 = 2$, $c_{1,2} = c_{2,1} = 10.0$

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 $\Lambda = 1$. Table 2 illustrates the stationary optimal policy when $k_1 = k_2 = 1$, for the same holding costs as in table 1.

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

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

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

service times. The optimal policy for $k_1 = 0$ is to take action $d = 2$ when $j_1 > 0$ and $j_2 < 2$ or when $j_1 = j_2 = 0$. If $k_1 = 2$, the optimal policy takes action $d = 1$ when $j_1 < 2$, or $j_1 = 2, 3, 4$ and $j_2 > 1$, or $j_1 > 4$ and $j_2 > 2$.

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 and table 2. Then, having monitored the system and estimated its parameters, the optimal policy could be obtained by a table look-up. This is feasible, but may consume quite a lot of storage.

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

4.1 Heuristic policies

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

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

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

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

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

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

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

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

These rules are based on approximating the effects of a switch. If j_1 jobs of type 1 are present and k_1 servers are available for them, then the average queue 1 increment during an interval of length x may be estimated as $x[\lambda_1 - \mu_1 \min(k_1, j_1)]$. Similarly for queue 2. Thus, a server is switched if that switch would help to

balance the holding costs, after taking account of its effect on the two queues. The above policy will be referred to as the 'heuristic'.

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

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

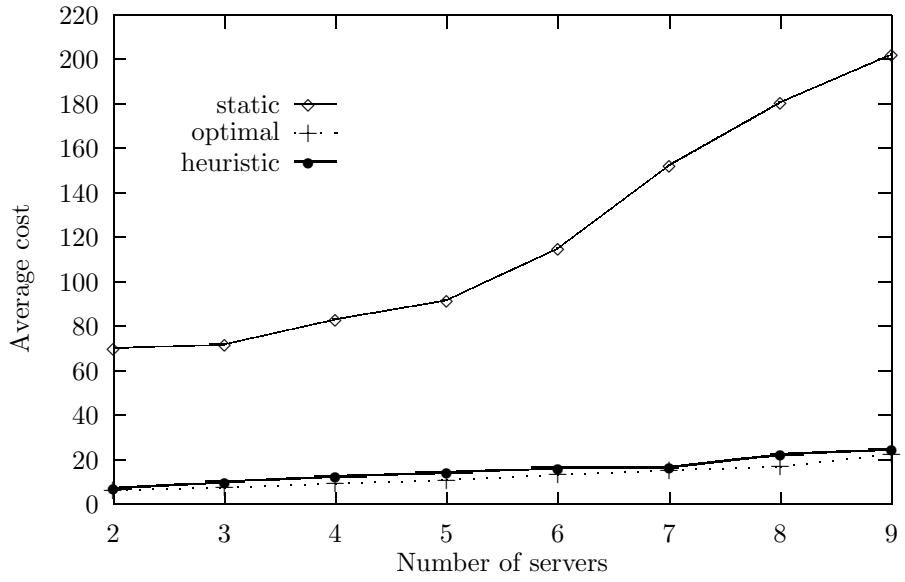

Figure 2: Policy comparisons: increasing N

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

A different comparison is illustrated in figure 3. Here the number of servers is fixed, $N = 4$, and the offered load increases, approaching saturation. The arrival rates are in the ratio 1:100 during phase 1 and 100:1 during phase 2, and are increased to produce the increase in total load.

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

In these experiments, 200000 job completions were simulated, and approxi-

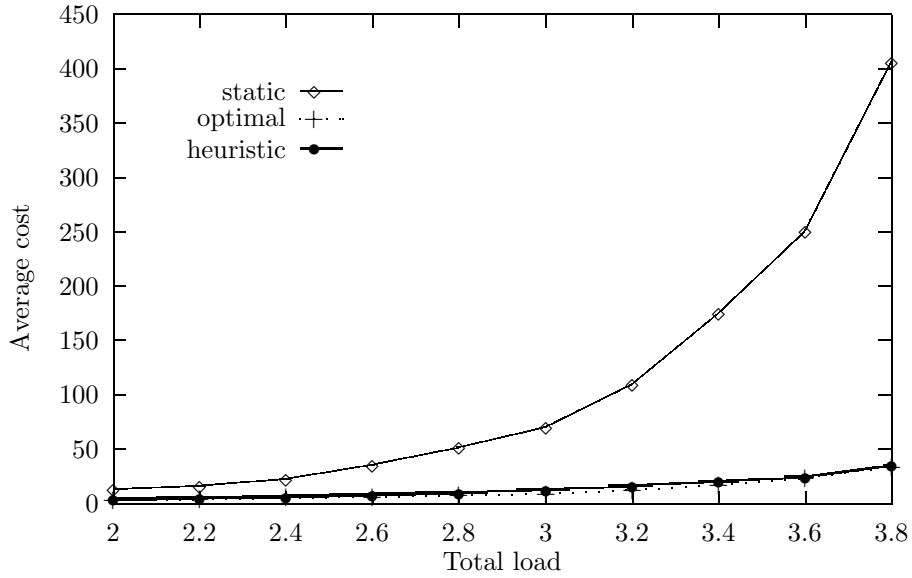

Figure 3: Policy comparisons: increasing loads

mately 1000 phase changes occurred. The longest simulation runs were for the optimal policy, because of the table look-ups.

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 2 and 3 suggest that its performance compares quite favourably with that of the optimal policy.

A natural generalization of this problem would be to consider more than two job types and clusters. That would lead to a significantly more complex model, which would be a worthy object of further study.

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] 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

[7] 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

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

[12] P. Whittle, *Optimisation over Time*, Vols 1 and 2, Wiley,