Semi-blind scheduling in a finite capacity system

Nigel Thomas[1], Jeremy T. Bradley[2], and William J. Knottenbelt[2]

[1] University of Newcastle, nigel.thomas@newcastle.ac.uk
[2] Imperial College London, {jb,wjk}@doc.ic.ac.uk

Abstract. In this paper a class of queueing model is studied where servers break down but the information regarding breakdowns is not immediately available. As such, jobs continue to arrive into a queue for some period after service has ceased. These queues have finite capacity, therefore there is also the possibility that queues become full causing job-loss. The effect of the duration of any delay on information propagation on the system response time and job loss is investigated and evaluated numerically.

1 Introduction

Models of systems where servers break down are of considerable theoretical and practical interest. However, most existing models assume that the presence of breakdowns is immediately known by any router or scheduler directing jobs to that queue. The only real exceptions are models where no rerouting of jobs takes place. This paper is motivated by Grid computing where scheduling will generally be performed remotely from servers. Unlike a cluster, the information used to inform a distributed scheduler cannot be constantly updated as the communication cost would be too great. As such, the scheduler makes routing choices based on longer term quantities. A small number of studies have been done on scheduling and resource management for Grid computing, [1, 6–8], but none of these studies deal with the consequences of failure at the resources.

In this paper the variable factor in routing is the operational state of the server, i.e. whether it is working or not. This presents the problem of how a scheduler can be informed of the changing states of servers. In the case of congestion, nodes can broadcast their problems to their neighbours causing jobs to be routed elsewhere (if possible). However, if a node has failed it is in no position to communicate its state and its neighbours must become aware of the problem passively. In general this takes time, but until the scheduler is informed they continue to act on old information. Clearly, a server which has just come into working order can send a message to the scheduler. Such active messaging can be assumed to carry a minimal overhead. However, a server that has just crashed cannot perform such an operation, therefore it is a matter for the scheduler, or some third party, to determine the working state of the server. Conventionally there are several mechanisms that can be employed for this purpose. For example the scheduler could regularly *ping* all the servers, or alternatively the servers could send regular *keep alive packets* to maintain a connection. However both these mechanisms carry an overhead which, by the regularity of the messaging, may be significant. Such messaging can be minimised by extending the intervals between messages; this however causes a delay between when a failure occurs and when the next communication takes place (or is expected). The length of the delay will be variable depending on the precise moment of failure relative to the expected time of the next message.

The aim of this paper therefore is to investigate the penalty of prolonged delays in information propagation to the scheduler. In a related context, Thomas and Mitrani [9] studied a class of queueing model where M/M/1 queues were arranged in parallel and the routing process could take account of the operational state of the servers without job loss. As expected the case where jobs are routed away from failures generally performs much better than cases where that information is ignored. Clearly this means that having that information is vital to good routing of jobs. However in practice it may take some time for the scheduler to become aware of failures. Mitrani and Wright [5] considered a model with the same structure but the effect of a server failure is to lose the entire contents of the associated queue. In this case the most important consideration is generally job loss. Thomas and Bradley [10] considered the same model as [9] but with finite capacity queues. Once again the routing decision is made independently of queue size, even if a queue is full, thus making job loss a possibility. The simpler and less applicable infinite queue case of the model presented here has been studied previously [11] and some initial work has been presented on dynamic analysis of a related model where failure notification is immediate [3].

2 Basic Model Definition

The general model under consideration in this paper can be described thus. Jobs arrive into the system in a Poisson stream with rate λ. There are N servers, each with an associated queue, to which incoming jobs may be directed. Each server goes through alternating independent operative and inoperative periods. While it is operative, the jobs in its queue receive service according to a given distribution and depart upon completion. When a server becomes inoperative (breaks down), the corresponding queue, including the job in service (if any), may remain in place or be lost entirely. The system model is illustrated in Figure 1.

Failures are considered to have two modes. In the first mode the server has failed, but the router is not aware of this and so jobs continue to arrive. In the second mode the router is aware of the failure and so no jobs are directed into this queue. The time it takes for the router to become aware of the failure can be modelled as a negative exponential delay with mean $1/\alpha$. However before this time has elapsed the server may have been repaired. Failures occur at instants separated by negative exponential delays with mean $1/\xi$, and are consequently repaired with time to repair negative exponentially distributed with mean $1/\eta$.

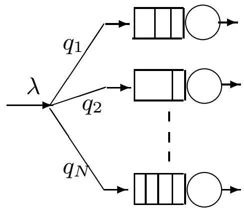

Fig. 1. A single source split among N unreliable nodes.

The operational state of the system is given as $\sigma = \{i(1), \ldots, i(N)\}$ where $i(j)$ is the operational state of an individual server, j. If, at the time of arrival, a new job finds the system in configuration σ, then it is directed to node j with probability $q_j(\sigma)$. These decisions are independent of each other, of past history and of the sizes of the various queues. Thus, a scheduling policy is defined by specifying 2^N vectors,

$$\mathbf{q}(\sigma) = [q_1(\sigma), q_2(\sigma), \ldots, q_N(\sigma)] \ \ , \ \ \sigma \subset \Omega_N \ \ ,$$

such that for every σ,

$$\sum_{j=1}^{N} q_j(\sigma) = 1 \ \ .$$

Each of the models presented here gives rise to a set of a Markov processes where the system state at time t is described by the pair $I(t), J(t) : t \geq 0$, where where $I(t) \subset \{0, \ldots, 2^N - 1\}$ represents the operational state of the system and $J(t)$ is the number of jobs in the queue being studied. Each queue may be studied in isolation as they exhibit a property referred to as *quasi-separability*, that is, the marginal distributions of the numbers of jobs in each queue are dependent only on the operational state of the system, $I(t)$, and not on the number of jobs in the other queues.

3 Finite Capacity Systems

In the basic model of finite capacity queues it is assumed that the scheduler does not know if a queue is full and that the consequence of sending a job to a full queue is that the job is lost. As well as the state where the server behaves normally (denoted by state 2), there are two states where no service occurs (states 1 and 0). In state 2, jobs arrive into the queue in a Poisson stream of rate λ and are served in FIFO order with service times negative exponentially distributed with mean $1/\mu$. In state 1 jobs arrive at the node in a Poisson stream rate λ but there is no service. In state 0 there are no arrivals and no service. The steady state probability of a given node, i, being in a given operational state is easily calculated:

$$P_i(2) = \frac{\eta}{\xi + \eta}$$

$$P_i(1) = \frac{\eta\xi}{(\alpha + \eta)(\xi + \eta)}$$

$$P_i(0) = \frac{\alpha\xi}{(\alpha + \eta)(\xi + \eta)}$$

On failure there are a number of different possibilities: the entire queue may be lost, the job in service may be lost, or the entire queue may be retained. In the case where the entire contents of the queue are lost on breakdown the consequences of information latency are the same as in the infinite case [11], namely that jobs will be lost from the system if they continue to be sent to a broken node. Hence, in the Markovian case, the job loss due to latency at node i is given by:

$$\overline{\lambda}_i \frac{\xi_i \alpha_i}{(\alpha_i + \eta_i)(\xi_i + \eta_i)}$$

where $\overline{\lambda}_i$ is the average arrival rate at at node i.

If the system is experiencing a heavy load then it is possible that redirecting jobs to alternative nodes may cause those queues to become full, thus causing job loss. Thus some of the jobs lost due to latency may have been lost from the system in any case, even if the latency $(1/\alpha)$ had been nil. Thus, under heavy load, reducing latency does not generally increase throughput. This point is further illustrated by numerical example in Section 4.

The case where the queue is preserved is much more interesting and is the main focus of this paper. In this case it is necessary to consider not just the increased response time due to jobs arriving when the server is broken, but also the potential job loss arising from the fact that the queue will fill up during broken periods. Therefore it is necessary to calculate the increase in the probability that the queue is full when the server is both broken and operative. This latter property is important since a recently repaired server will be suffering under a backlog of jobs, making it more likely that arriving jobs find the queue full.

This suggests two potential strategies for minimising the job loss due to information latency:

1. If the queue size passes a certain threshold then the node signals the scheduler to send fewer jobs (or none at all). This is similar to a conventional quench packet used to reduce congestion problems in packet switched networks. The result of this strategy is that the queue is less likely to become full during repair, consequently the job loss probability should be reduced.
2. When the server repairs there is a delay before the scheduler restarts sending jobs to that node. This may be achieved by a simple time delay, or alternatively the node may signal the scheduler once the queue size has fallen below a certain threshold.

A scheduler working in such an environment may be said to be *semi-blind*, as it is basing routing decisions on only partial, and possibly out of date, queue status information.

These strategies are likely to work well when the system load is relatively light. However, when the load is high there is a clear cost to sending more jobs to the remaining (operative) nodes and this may cause more jobs to be lost elsewhere. If all nodes are heavily loaded then it is impossible for the scheduler to reroute jobs away from busy nodes, and so the strategy will fail. Clearly these strategies are going to add additional states to the scheduler behaviour. If each node operates a single threshold then the operational state space grows from 3^N to 6^N, since it will be necessary to track the queue size relative to the threshold in all modes for all servers. In addition such a feature would destroy the quasi-separability condition used to decompose the model solution in Section 3. A simple stochastic delay on repair prior to sending jobs would increase the operational state space from 3^N to 4^N, but crucially preserve the quasi-separable nature of the model making it much less costly to solve.

There is also the problem of determining the optimal value for the queue size thresholds and scheduler delay for a given set of system parameters. It is reasonable to assume that failures are rare enough that the system will tend to a steady state during operative periods. Thus the distribution of the number of jobs at node j with a capacity K_j at time of failure will tend towards the steady state solution for a simple $M/M/1/K_j$ queue. In addition the average number of jobs arriving at a broken node will be $\lambda_j/(\eta_j + \alpha_j)$, where λ_j is the average arrival rate at node j when it is not known to be broken. It is not difficult therefore to set the threshold T_j, such that the average job loss at node k will be known. However it is much more difficult to calculate the resultant job loss at other nodes as a function of all the T_j's, $j = 1, \ldots, N$.

4 Numerical evaluation

Numerical experiments have been carried out using the PEPA Workbench [2], a stochastic process algebra tool which is useful for defining complex behaviour, such as that of the scheduler here. Initial numerical experiments have focused on two key performance measures, the average rate of job loss and the average response time. All the figures show results for a two identical node system with relatively small capacity queues, $N = 9$. In future experiments we will consider larger models and additional performance measures, such as response time quantiles, not directly available from the PEPA Workbench.

Figures 2 and 3 show results of job loss varied against the latency, α, for the basic model without threshold levels. In Figure 2 the job loss resulting from rerouting after the exponentially distributed delay is compared with the job loss when no rerouting takes place (which does not vary with α). As expected at low values of α (long delay) the job loss tends to the maximum value attained when no rerouting takes place. The reason for this is obviously that as the delay increases, the probability of the broken node becoming full also increases.

However, as α increases towards instant rerouting ($\alpha \rightarrow \infty$) there is not a uniform decrease in job loss, but rather the rate initially decreases as expected, but then rises again, the minima in this case being around $\alpha = 64$. The reason for this is not quite so intuitive, but is caused by the fact that rerouting from a broken node will cause a greater load at the working node. This will cause an increase in the full probability at that node, and hence an increase in job loss from that node. At low load this effect will be minimal and fast rerouting will be near-optimal for reducing job loss. However, as system load increases this effect will become more apparent and it will become necessary for the scheduler to use the entire queueing capacity of the system to reduce job loss. In this model the only mechanism available to the scheduler is to wait longer before acting on the failure (we have modelled this as the delay before the scheduler knows of the failure). Therefore in order to utilise the queue capacity at the broken node the minimum job loss is reached by delaying the rerouting of jobs.

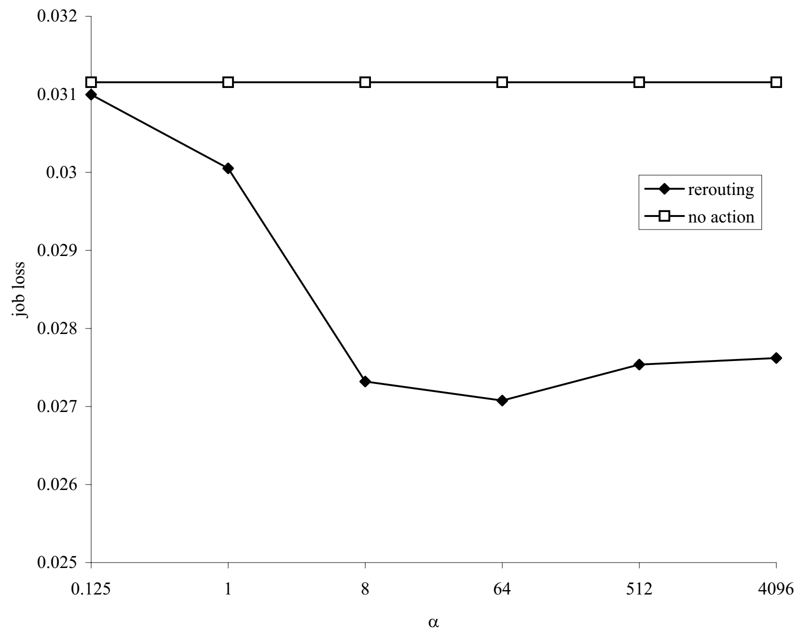

Fig. 2. Job loss varied against latency $\mu_i = 10, \eta_i = 10, \xi_i = 1, \lambda = 10, q_i = 0.5$

This is illustrated further in Figure 3, where results are shown for different repair and failure rates when the load is somewhat higher. In each case the probability of being working or broken is the same. As expected the longer the repair periods the greater the probability that the working queue will become full, hence increased job loss. The implication of this is that the system becomes more reliant on the capacity of the broken queue. Hence, as η decreases, so does the optimal value of α to minimise job loss. That is, the longer the repair period,

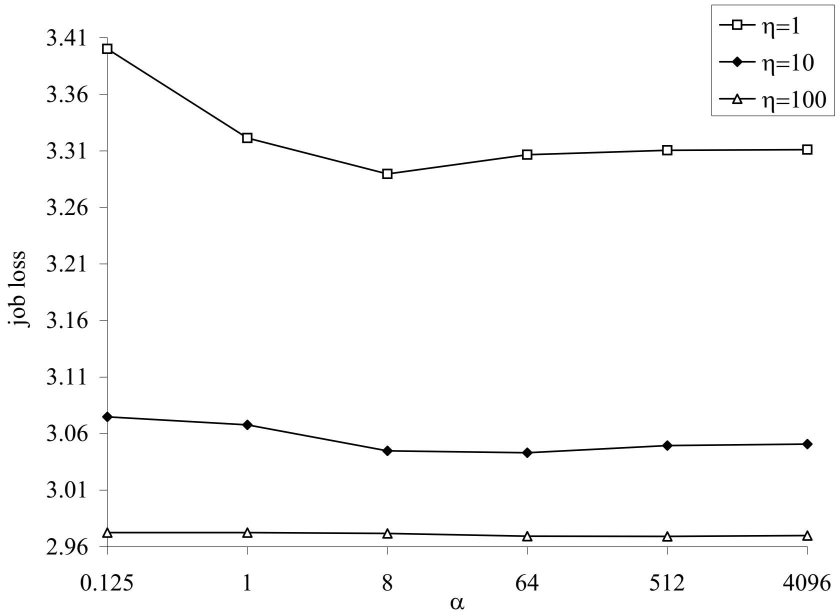

Fig. 3. Job loss varied against latency $\mu_i = 10, \xi_i = \eta_i/10, \lambda = 20, q_i = 0.5$

the longer the scheduler will continue to send jobs to a broken queue in order to reduce the job loss from the system as a whole.

Figure 4 shows the average response time for successful jobs for the same parameters as Figure 3. The value of average response time is greatly influenced by the rate of job loss. Thus, although we would expect instances where the latency is large (α is small) to give poor response time, this is not universally true because the job loss is also greater in these cases and hence there are fewer jobs entering the queues (this explains the dip at $\alpha = 0.125$ when $\eta = 1$).

It is clear from Figure 4 that when the repair rate is relatively fast, the latency has little effect on response time. This has been observed in earlier studies, e.g. [10], where the conclusion was that in cases of rapid repair there is little advantage to be gained from rerouting. As the repair rate decreases there is an increasing effect from latency; despite the fact that fewer jobs are successful, those that are are still experiencing a poorer performance than when the latency is reduced. However, comparing Figures 3 and 4 shows that there is a clear trade off between gains in response time and job loss. In absolute terms increasing the average latency from 1/4096 ($\alpha = 4096$) to 1/8 ($\alpha = 8$) increases the average response time by 1.5% with a decrease in the rate of job loss of 0.2%. Ultimately the optimal delay will therefore depend on whether reliability or performance is the more important factor.

In Figure 5 the model with a threshold level is introduced. The aim here is to limit the number of jobs in a queue without adding an excessive communication overhead. The threshold prevents one queue from becoming too full if the other

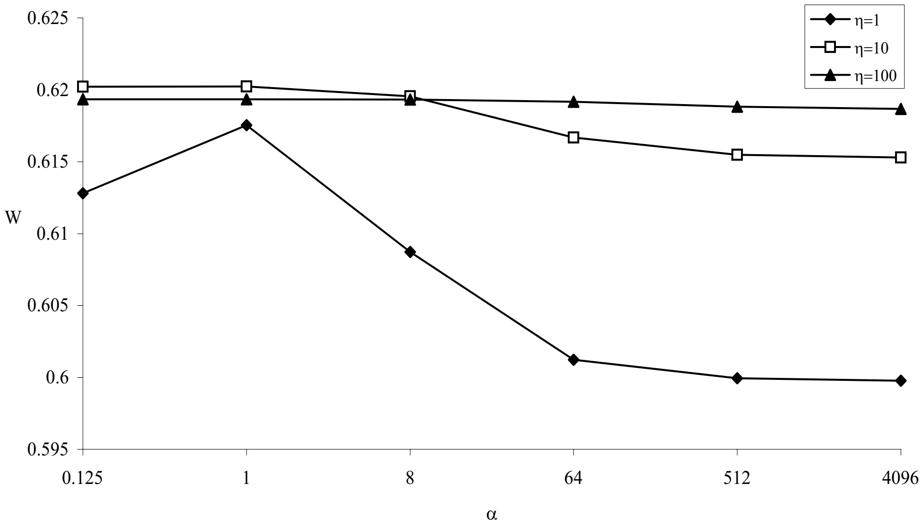

Fig. 4. Average response time varied against latency $\mu_i = 10, \xi_i = \eta_i/10, \lambda = 20, q_i = 0.5$

has fewer jobs than the threshold. Thus the queues can only ever become full (and cause job loss) if the other queue is either above the threshold or broken. The rate of job loss shown in Figure 5 is far lower than that for the same paremeter set shown in Figure 3, a clear indication of the success of this strategy. However,

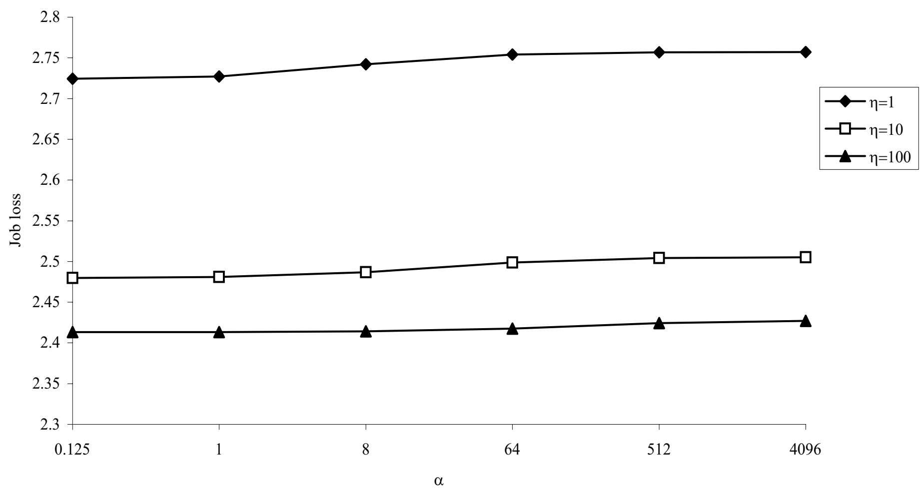

Fig. 5. Job loss varied against latency, $N = 9$, threshold=6, $\mu_i = 10, \xi_i = \eta_i/10, \lambda = 20, q_i = 0.5$

it is necessary to set the threshold sufficiently high that both queues only exceed it together as rarely as possible. This is difficult to achieve if the load is high. In Figure 5 the load is high (demand exceeds the service capacity) and this causes a the rerouting of jobs away from broken nodes to be less optimal. There is a slight advantage in this case towards only rerouting away from queues above the threshold and ignoring the operative state of the server. This is because all jobs will be directed to one node if the other is known to be broken, regardless of its queue size.

Figure 6 shows the average response time for the same parameter set as Figure 5. Since the job loss is more stable in this case, it has less of an impact than in Figure 4, hence the system has more of the characteristics of the simpler infinite queue system. There is a clear penalty for long latency, particularly when the repair times are long. Hence for response time, unlike job loss, there is a clear advantage in rerouting away from failures quickly when the threshold is utilised.

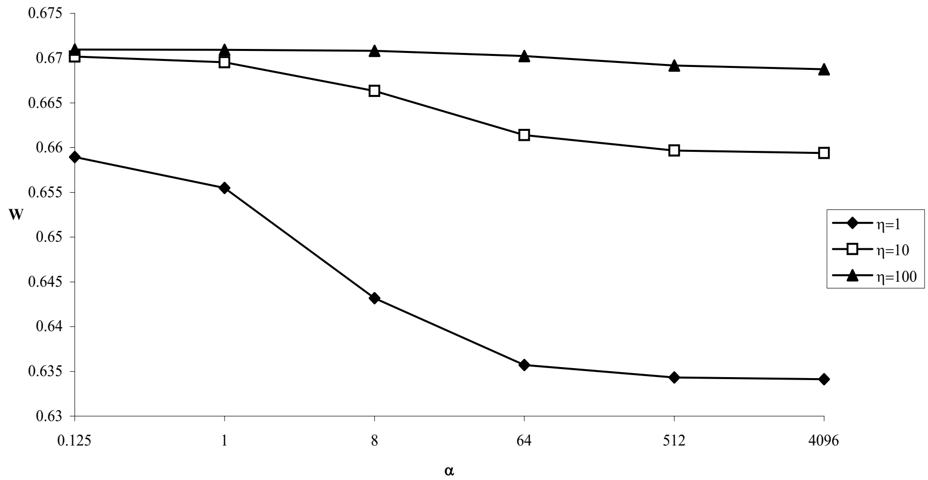

Fig. 6. Average response time varied against latency, $N = 9$, threshold=6, $\mu_i = 10, \xi_i = \eta_i/10, \lambda = 20, q_i = 0.5$

5 Conclusions

It is clear from the results presented here that information latency can have a profound effect on system performance and reliability. In addition it is the case that the current Grid implementations have large latencies and very few services are being constructed with a fault tolerant perspective. It is evident therefore that unreliable services may well give rise to very poor performance, even when alternative services could be employed.

The results presented here are not all intuitive and differ from the simpler infinite queue case. The effect of job loss is marked and it is shown that in some cases it is better not to act immediately (or at all) to route away from failures. For finite capacity queues two simple, low cost, strategies have been suggested to reduce the impact of failures. The implementation of these strategies over a network of possibly interdependent services suggests a simple game problem may be constructed to optimise the scheduler. Such a game could, potentially, involve pricing differences to represent urgent requests or requests to currently unavailable services.

References

1. J. Cao, S.A. Jarvis and S. Saini, ARMS: An agent-based resource management system for grid computing, *Scientific Programming*, **10**(2), 2002.
2. G. Clarke, S. Gilmore, J. Hillston and N. Thomas, Experiences with the PEPA Workbench modelling tools, *IEE Proceedings - Software*, **146**(1), 1999.
3. S. Martin and I. Mitrani, Optimal Scheduling Among Intermittently Unavailable Servers, in: *Proceedings of the first Workshop on Grid Performability Modelling and Measurement*, National eScience Centre, March 2003.
4. I. Mitrani and R. Chakka, Spectral Expansion Solution for a Class of Markov Models: Application and Comparison with the Matrix-Geometric Method, *Performance Evaluation*, **23**, pp. 241-260, 1995.
5. I. Mitrani and P.E. Wright, Routing in the Presence of Breakdowns, Performance Evaluation, 20(1-3), pp. 151-164, 1994.
6. B. Nitzberg and J.M. Schopf, Current Activities in the Scheduling and Resource Management Area of the Global Grid Forum, in: D.G. Feitelson, L. Rudolph and U. Schwiegelshohn, *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science 2537, Springer Verlag 2002.
7. J.M. Schopf, A General Architecture for Scheduling on the Grid, *Argonne National Laboratory preprint*, ANL/MCS-P1000-1002, 2002.
8. D.P. Spooner, S.A. Jarvis, J. Cao, S. Saini and G.R. Nudd, Local grid scheduling techniques using performance prediction, *IEE Proceedings Computers and Digital Techniques*, **150**(2), pp. 87-96, 2003.
9. N. Thomas and I. Mitrani, Routing Among Different Nodes Where Servers Break Down Without Losing Jobs, in: *Quantitative Methods in Parallel Systems*, Springer-Verlag, pp. 248-261, 1995.
10. N. Thomas and J. Bradley, Decomposing models of parallel queues, in: Proceedings of the 4th International Workshop on Queueing Networks with Finite Capacity, Ilkley, July, 2000.
11. N. Thomas, The effect of information latency on performance, in: Proceedings of the 19th UK Performance Engineering Workshop, University of Warwick, July, 2003.