Stochastic analysis of scheduling strategies in a Grid-based resource model

N. Thomas, J.T. Bradley and W.J. Knottenbelt

Abstract: A model inspired by a scenario found in Grid-based scheduling systems is considered. Scheduling is performed remotely without access to up-to-date resource availability and usage information. This system is modelled as a collection of queues where servers break down and are subsequently repaired. There is a delay before the scheduler learns of failures, and requests may continue to arrive into a resource queue for some time after active service has ceased. The queues are considered to be persistent under failure. However, these queues have finite capacity; therefore there is the possibility that queues become full, causing job-loss. Stochastic process algebra and stochastic probes are used to analyse this model to find steady-state measures and passage time distributions. 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 a 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 [1–4], but none of these studies deals with the consequences of failure at the resources.

In this paper, the variable factor in routing is the operational state of the server, that is, 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 and, 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 to determine server availability. One such mechanism, as employed in SOAP, for example, is that when a job is submitted to a node the scheduler will expect an acknowledgment. If no acknowledgment arrives during a given time frame (generally in the order of tens of seconds), it is assumed that the request has failed and an exception is raised. However, the failure to receive an acknowledgment is no guarantee that a service has failed and so some additional communication is still required to determine the status of the node. Additional mechanisms include the scheduler regularly *ping*ing all the nodes, or alternatively the nodes can send regular *keep alive packets* to maintain a connection. 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 is to investigate the penalty of prolonged delays in information propagation to the scheduler. In a related context, Thomas and Mitrani [5] 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. Mitrani and Wright [6] 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 [7] considered the same model as [5] 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

IEE Proceedings online no. 20041091

doi: 10.1049/ip-sen:20041091

Paper received 24th August 2004

N. Thomas is with the School of Computing Science, University of Newcastle Upon Tyne, Newcastle Upon Tyne, NE1 7RU, UK

J.T. Bradley and W.J. Knottenbelt are with the Department of Computing, Imperial College London, Huxley Building, 180 Queen's Gate, London SW7 2BZ, UK

previously [8] and some initial work has been presented on dynamic analysis of a related model where failure notification is immediate [9].

2 Scheduling and brokerage in Grid architectures

There are many approaches to scheduling of tasks performed within Grid systems. Some approaches derived from traditional distributed systems rely on having almost complete knowledge of the resources available. Thus jobs may be directed to the node which will be able to complete the task first with some degree of certainty. If problems arise, for instance a node fails or a task takes much longer than predicted, then a new schedule can be computed relatively easily. An example of such a system is [4] where local (cluster) scheduling has been developed for Grid-enabled jobs using a fast genetic algorithm to compute near-optimal schedules across highly dependent distributed tasks. Such an approach is highly effective on a local level, but cannot be directly applied to wider scheduling issues on the Grid as it is generally not feasible to know the status of resources sited remotely from the scheduler, particularly as those resources are subject to local rescheduling.

The general problem encountered here is referred to as *brokering*, that is, incorporating scheduling decisions over resources across multiple domains [3]. A broker operating on the Grid must be able to select distributed resources over which it has no control and information about which is often limited or stale [10]. The ARMS system [1] attempts to overcome this problem by using agents representing local domains to negotiate prior to allocation of local resources. Other approaches rely on some centralised monitoring system, such as MDS [11], to collect up-to-date information on available services. Such approaches potentially suffer from the obvious problems that the information needs to be kept up-to-date and also that additional layers of communication are introduced that themselves have a negative impact on performance. Examples of such systems include Condor-G [12], Nimrod/G [13] and AppLeS [14]. These approaches are best suited to very large tasks where the computation times greatly exceed the additional overhead and the penalty for a poor decision is large; however, they still have to cope with old, and possibly inaccurate, information.

To work around the problem of out-of-date resource information, several systems [14, 13, 10] employ resource reservation to negotiate some kind of service level agreement prior to deployment. This requires direct communication between the scheduler and resource managers to determine the availability, and possibly usage cost, of resources. Once a good selection has been made the reservation can be confirmed. However, time has elapsed since the first enquiry and so the resource status may have changed, causing the performance to be less than expected. In addition, a resource manager is unlikely to be passive during this period and may also be offering services to other schedulers. This brings the two distinct viewpoints, the *system-centric* resource managers and the *user-centric* schedulers, into direct competition. The system therefore has inherently complex interactive behaviour and ultimately will be optimised to predominantly suit one or other view, but not both.

3 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 Fig. 1.

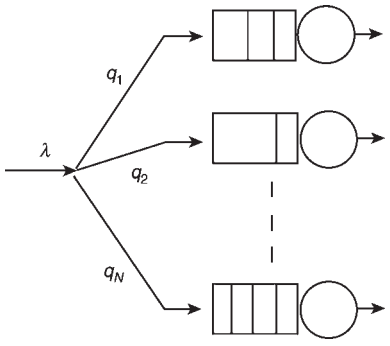

Fig. 1 *Single source split among N unreliable nodes*

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 its 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 $1/\eta$.

The operational state of the system is given as $\sigma = \{i(1), \ldots, i(N)\}$, where $i(j) \in \{0, \ldots, M-1\}$ given M distinct operational states 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 M^N vectors, $\mathbf{q}(\sigma) = [q_1(\sigma), q_2(\sigma), \ldots, q_N(\sigma)], \sigma \subset \Omega_N$, such that for every σ, $\sum_{j=1}^{N} q_j(\sigma) = 1$.

Here, we consider models with $M = 3$ distinct operational states. 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 in the queue in a Poisson stream with 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 with rate λ (since the scheduler is unaware of the server failure) but there is no service. In state 0, there are no arrivals (since the scheduler is aware of the server failure) and no service.

Each of the models presented gives rise to a set of Markov processes where the system state at time t is described by the pair $I(t), J(t) : t \geq 0$, where $I(t) \subset \{0, \ldots, 3^N - 1\}$ represents the known 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.

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. The steady-state probability of a given node i being in a particular operational state is calculated as

$$P_i(0) = \frac{\alpha\xi}{(\alpha+\eta)(\xi+\eta)} \quad P_i(1) = \frac{\eta\xi}{(\alpha+\eta)(\xi+\eta)}$$
$$P_i(2) = \frac{\eta}{\xi+\eta}$$

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 [8], namely that jobs will be lost from the system if they continue to be sent to a broken node. Hence, in this case, the job loss due to latency at node i is given by

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

where $\bar{\lambda}_i$ 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 the numerical examples 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 the following strategy for minimising the job loss due to information latency. 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. 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.

This strategy is likely to work well when the system load is relatively light. However, when the load is high and split between fewer operative nodes, this may cause a higher job loss rate in the system. 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 this strategy is 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 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 $T_j, 1 \leq j \leq N$.

4 Numerical evaluation

Experiments have been carried out using both the PEPA Workbench [15] and `ipc` [16] stochastic process algebra tools, which are useful for analysing complex behaviour, such as that of the scheduler. All the figures show results for a two identical node system with relatively small capacity queues, $N = 12$. Parameters in the following figures are given in terms of rate per hour. The average service time is 10 min $(\mu = 6)$, the average inter-arrival time is 6 min $(\lambda = 10)$ and the availability of each node is fixed as 99.01%. Other parameters are varied as indicated.

Numerical results are derived for steady-state and response time distributions. Steady-state results are derived conventionally and are focused on two key performance measures: the average rate of job loss and the average response time. Passage time distributions are derived by means of *stochastic probes*, described below.

4.1 Stochastic probes

A stochastic probe is a fragment of stochastic process algebra, for our purposes PEPA, which describes the start and end points of a measurement that we wish to make on a system. The start and ends points are specified by the modeller in terms of the behaviour that the model exhibits: for example, below we generate results that look at the time taken between a node-failure event and a queue-full event.

Stochastic probes [17] are used to define specific response time measurements on the semi-blind scheduling system. Briefly, a stochastic probe is defined by a regular expression. In this case, however, the atoms of the regular expression are action names drawn from the alphabet of the underlying process algebra model. The probe is automatically translated into a component of the stochastic process algebra and composed with the original model in order to make the required measurement.

The power of using a stochastic probe over a stochastic process algebra lies in the fact that:

- its uses the native action-based (rather than state-based) language of the process algebra
- it can be used to define arbitrary behaviour before a measurement is started or stopped
- it is designed not to interfere in any behavioural or temporal way with the model it is measuring.

We make use of the Imperial PEPA Compiler, `ipc`, for processing the stochastic probes and the HYDRA/DNAmaca Markov chain analyser [18] for producing the response time results.

4.2 Results

Figures 2 and 3 show results of the average rate of job loss varied against the latency, α, for the basic model without threshold levels. In Fig. 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). In addition we also show the job loss arising from a Markov modulated arrival process which is more bursty than the Poisson process. This Markov modulated process has two arrival rates, $\lambda_{high} = 15$ and $\lambda_{low} = 5$, with a rate of switching between them of 1 (per hour), the process spending an equal proportion of time in each state on average. 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. The shape of the plots for Poisson and Markov modulated arrivals is nearly identical, although with the more bursty process giving a higher rate of job loss. This plot and other such experiments give us some confidence that the Poisson assumption made in this model does not generally affect the conclusions drawn from this study, although the absolute measures may be optimistic.

Figure 3 shows results for different repair and failure rates, although in each case the probability of being working or broken is the same. The longer the repair periods the greater is the probability that the working queue will become full, and hence lead to increased job loss. The implication of this is that the system becomes more reliant on the capacity of the broken queue. Hence, as η_i decreases, so does the optimal value of α to minimise job loss. That is, the longer the repair period, 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.

As α increases towards instant rerouting ($\alpha \to \infty$) there is not a uniform decrease in job loss, but rather the rate initially decreases as expected, but then rises slightly again, the minima in this case being around $\alpha = 1$ for $\eta_i = 0.1$ and $\eta_i = 1$. 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, to utilise the queue capacity at the broken node the minimum job loss is reached by delaying the rerouting of jobs.

Figure 4 shows the average response time for successful jobs for the same parameters as Fig. 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 also explains why the plot of $\eta_i = 0.1$ appears to have potentially better performance than $\eta_i = 10$). It is clear that when the repair rate is relatively fast, the latency has little effect on response time. This has been observed in earlier studies, for example in [7], 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. It is interesting to note that the optimal response time occurs when $\alpha = 4096$ and $\eta_i = 0.1$, that is when the repair period is longest and the rerouting is fastest. This is because in this scenario there is the greatest chance that a job will be directed to a working node and that node will still be operational when the job reaches the head of the queue. Comparison of Figs. 2 and 4

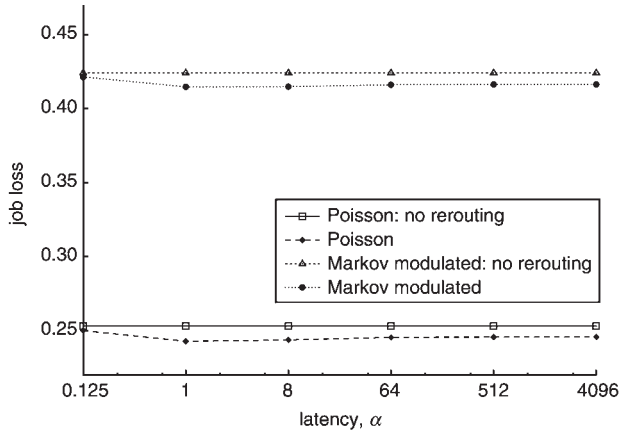

Fig. 2 *Mean rate of job loss varied against latency*
$\mu_i = 6$, $\eta_i = 1$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$

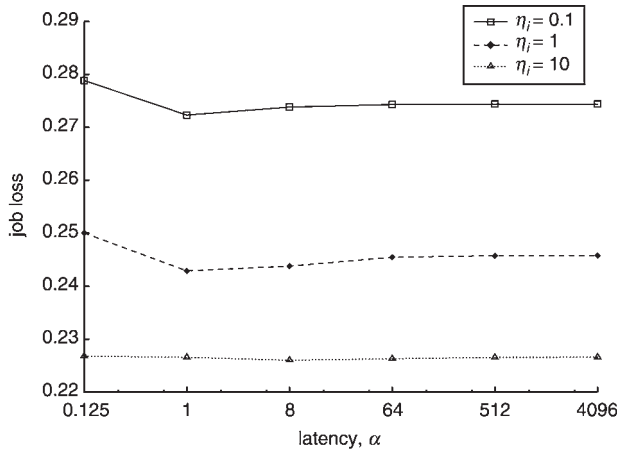

Fig. 3 *Mean rate of job loss varied against latency*
$\mu_i = 6$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$

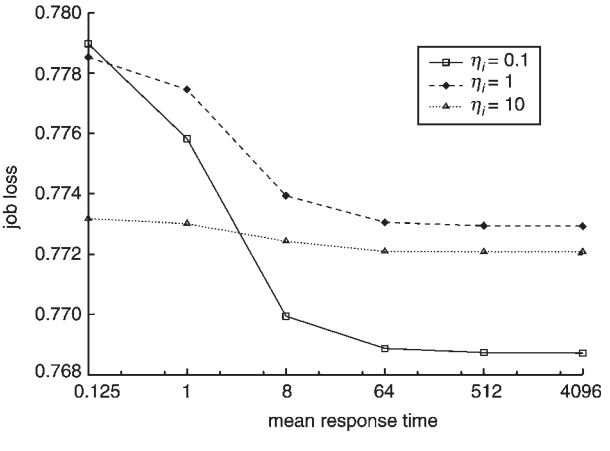

Fig. 4 *Average response time varied against latency*
$\mu_i = 6$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$

shows that there is a potential tradeoff between gains in response time and job loss, with the relative minima occurring in different regions of the plots. Ultimately the optimal delay will therefore depend on whether reliability or performance is the more important factor.

Figures 5 and 6 show the response time density and cumulative distribution function for the time elapsing following a failure at a node until the queue at that node becomes full. Figures 7 and 8 show the corresponding functions for the time elapsing following a repair at a node until the queue at that node becomes full. In each case the elapsed time may include the node being repaired or failing again prior to becoming full. Comparing these four plots it is evident that a queue is no more likely to become full following a failure at that node than following a repair. This might appear counterintuitive; however, when a repair occurs the queue is likely to contain a significant number of jobs due to receiving some during the failed period. Therefore initially it is prone to becoming full until the backlog is cleared. The probability of becoming full drops off relatively quickly in this case. Following a failure the probability density of the queue becoming full drops off slightly less quickly as not only may jobs continue to arrive if the scheduler has not been updated, but also the other node may fail (or may already be inoperative). In addition a repair action will happen at some point following a repair and at this time jobs will be directed back to that node regardless of the number already in the queue. More surprising is the apparent lack of distinction between the plots for the various repair rates considered, unlike the steady-state results. Figure 5 shows the greatest distinction in this regard, particularly at the start of the plot where the faster repair time shows a more rapid decline in probability density than would be expected. As time increases, however, the slower repair time (and hence longer operative period) becomes more advantageous as subsequent failures are much less likely to occur in this time frame.

Figure 9 shows response time distributions for an arbitrary successfully completing job from the time it enters the system to when it completes its service. The response time is made up of the service of this job, plus the services of all the jobs ahead of it in the queue (if any) and any repair periods that may occur before successful completion. Because no jobs are lost once they enter the queue it is only necessary to keep track of the number of service actions from an arrival event in order to compute the response time. The graph shows multiple response times in conditions of underload and overload, together with an Erlang(12) distribution. The Erlang distribution represents the limiting case where there are no server failures and the system is always completely full, so an incoming job sees 12 successive exponential service times.

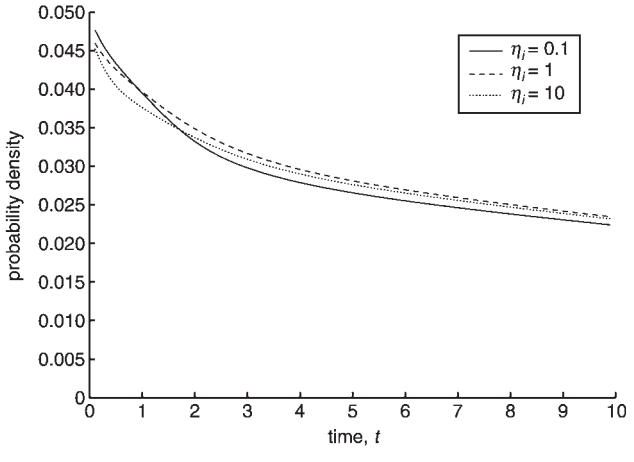

Fig. 5 *No threshold model: response time density of time taken from a failure to a full queue event for different values of* η_i
$\mu_i = 6$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$, $\alpha = 1$

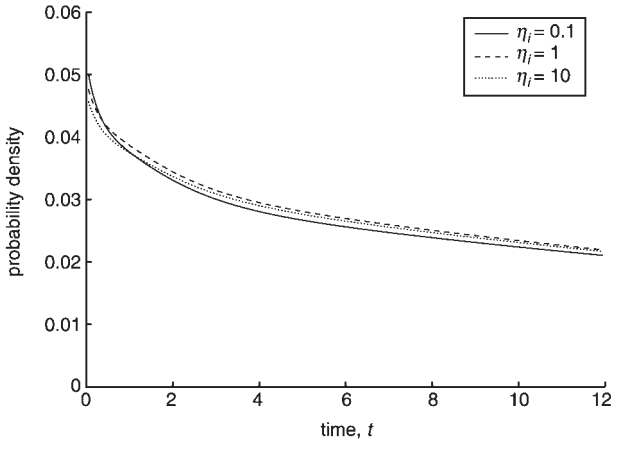

Fig. 7 *No threshold model: response time density of time taken from a repair to a full queue event for different values of* η_i
$\mu_i = 6$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$, $\alpha = 1$

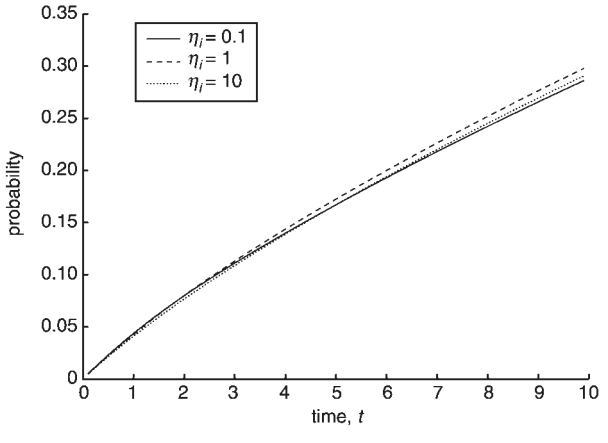

Fig. 6 *No threshold model: cumulative response time from a failure to a full queue event for different values of* η_i
$\mu_i = 6$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$, $\alpha = 1$

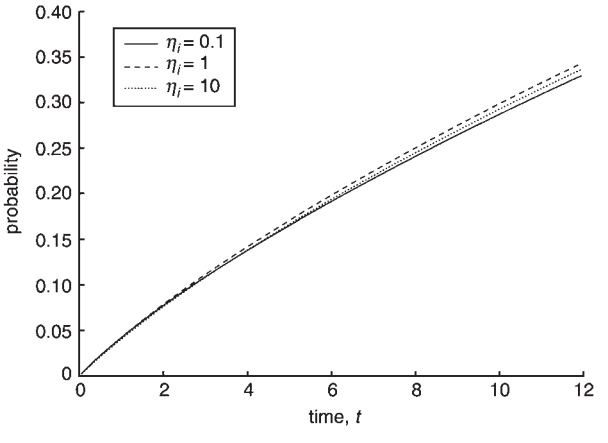

Fig. 8 *No threshold model: cumulative response time from a repair to a full queue event for different values of* η_i
$\mu_i = 6$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$, $\alpha = 1$

In Fig. 10, 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 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 Fig. 10 is far lower than that for the same parameter set shown in Fig. 3 – a clear indication of the success of this strategy.

However, 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 Fig. 10, the load is fairly high and this causes rerouting of jobs away from broken nodes to be less optimal. There may be a potential advantage in this case towards only rerouting away from queues above the threshold and ignoring the operative state of the server, particularly when the repair rate is relatively fast. 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 11 shows the average response time for the same parameter set as Fig. 10. Since the job loss is more stable in this case, it has less of an impact than in Fig. 4; hence the system has more of the characteristics of the simpler infinite queue system. There is a large penalty for long latency, particularly when the repair times are long. Hence for response time there is an advantage in rerouting away from failures quickly when the threshold is utilised.

Figures 12 and 13 show the response time density and cumulative distribution functions for the elapsed time from a failure of a node to that queue becoming full. The first point to observe is that the probability density is much heavier tailed than in the case not employing a threshold, i.e. there is a much lower chance that a job will be lost as

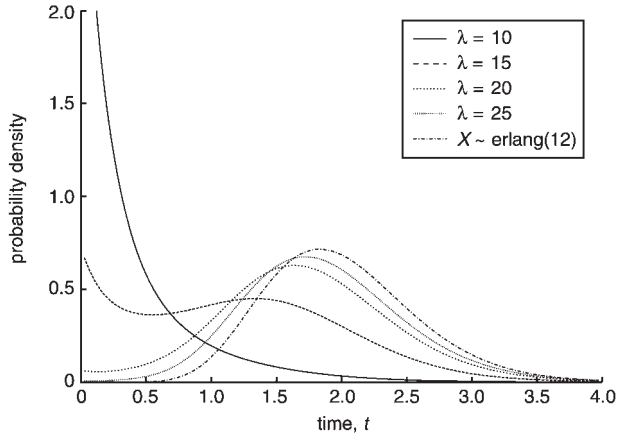

Fig. 9 *Total response time for a completing job; shown for increasing input job rate, λ, for conditions of underload to overload*

$\mu_i = 6$, $\eta_i = 1$, $\alpha = 1$, $\xi_i = \eta_i/100$, $q_i = 0.5$

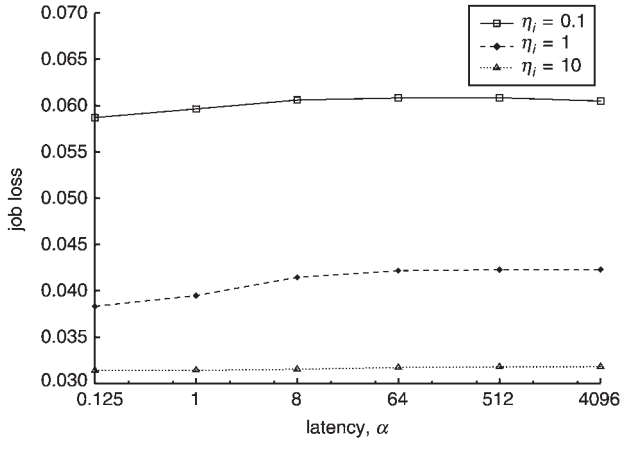

Fig. 10 *Job loss varied against latency*

$N = 12$, threshold $= 9$, $\mu_i = 6$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$

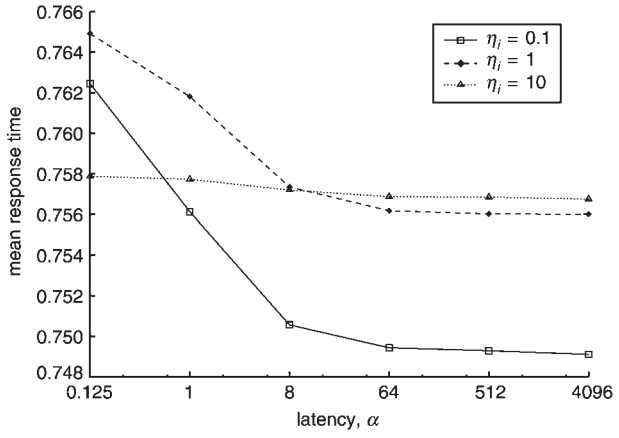

Fig. 11 *Average response time varied against latency*

$N = 12$, threshold $= 9$, $\mu_i = 6$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$

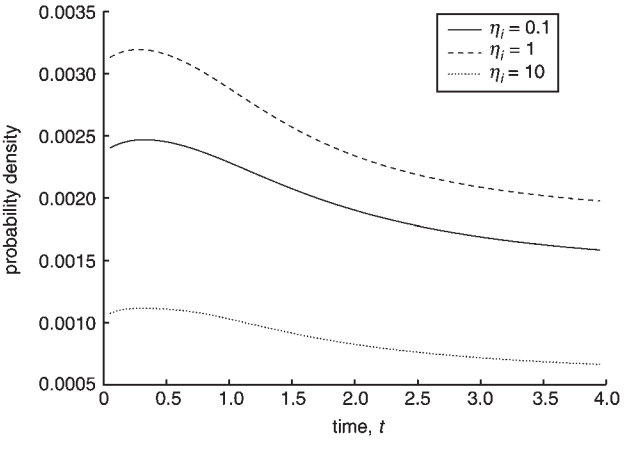

Fig. 12 *Threshold model: response time density of time taken from a failure to a full queue event for different values of η_i*

$\mu_i = 6$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$, $\alpha = 1$

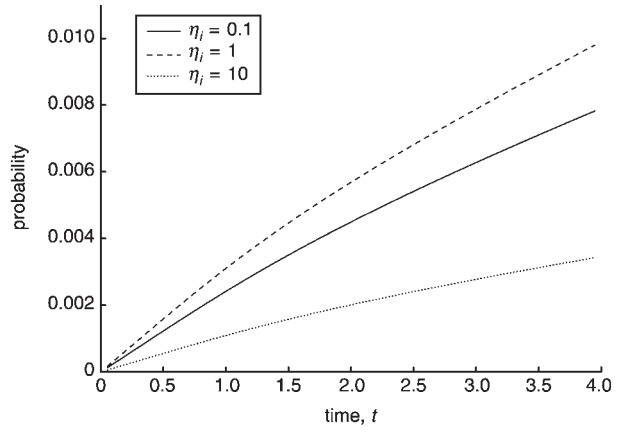

Fig. 13 *Threshold model: cumulative response time from a failure to a full queue event for different values of η_i*

$\mu_i = 6$, $\xi_i = \eta_i/100$, $\lambda = 10$, $q_i = 0.5$, $\alpha = 1$

a direct result of a failure than in the equivalent nonthreshold case. Thus we can conclude that the queue is much more stable with respect to job loss when a threshold strategy is adopted. It is curious to note that the plots of $\eta_i = 1$ show worse performance than for $\eta_i = 0.1$; the same phenomenon is evident in Fig. 3 at $\alpha = 1$. This is because, when $\alpha = 1$, proportionally very few jobs arriving during a repair period will be directed to the broken node if $\eta_i = 0.1$, whereas if $\eta_i = 1$ far more jobs arriving during a repair period will be directed to the broken node. If the repair period is much shorter then the effect of a service interruption will be much less significant on response time.

5 Model limitations

There are a number of practical limitations to the model which are presented here:

Resource reservation: Resource reservation is one mechanism that is employed to increase predictable performance in Grid systems. However, at present, the only practical evaluations that have been made are based on systems with a relatively low load; hence it is always possible to find sufficient available resources. Under a high load scenario, resource reservation is insufficient in itself to provide any performance guarantee as it will become necessary to either make future reservations or delay reservation until such time as resources are known to become available. These additional mechanisms introduce several possibilities for unpredictable performance problems, not least of which is that the timeliness of information will be crucial. The model presented here does not consider resource reservation.

Distributed scheduling: The model presented here considers a single scheduler which is remote from the services it uses. In reality, there will be many such schedulers which are at, or close to, the sources of the tasks. Modelling these multiple schedulers as one entity is not a significant problem; but it does overlook the issue that different schedulers may be operating on different information.

Multiple services: It is important to note that the model here considers each of the nodes to be independent whilst offering equivalent atomic service. In reality, this unlikely to be true. Most services are compound, meaning that any given service will itself request other services to perform some or all of its functionality. This is not a significant direct challenge to the atomic assumption made in the model because the service time can include many individual, possibly distributed services, although higher moments may be lost.

Common mode failure: For a specialist service, it is possible, likely even, that two apparently distinct services utilise the same service to perform some operation. Whilst this may be considered not to be significant for performance, it does have an important effect on reliability. Specifically, if this shared service fails then both the parent services will fail. Thus, under certain circumstances, the operational state of the nodes may not be independent as stated in the model. Furthermore this information will not be available to the scheduler as the nodes will only present information pertaining to the parent services and not any services they may employ. To overcome this type of hierarchical modelling problem, greater specific detail about the scheduling system has to be gathered and tools such as SPAs (as used here) and LQNs (layered queueing networks) [19] used for further investigation.

Degrading service: In this paper, we consider only that a node is either working or not. In reality each node may contain many servers and so a single failure may only partly degrade the service. When the system is lightly loaded this degradation may be very slight indeed. The assumption made here is that we are only really interested in catastrophic failures causing the complete loss of a service, although we accept that degrading service is an interesting problem and one which can affect scheduling significantly.

6 Conclusions

We have studied scheduling strategies that are of use when minimising job loss and improving reliability and fault-tolerance in Grid systems. We have empirically measured job loss and response times across these different strategies and shown that a system of thresholding within the resource buffers can dramatically improve buffer utilisation and overall fault-tolerance.

In all strategies, it is clear that profound changes in system performance and reliability can be achieved by modifying the information latency as seen by the scheduler. In addition, it is the case that current Grid implementations have large latencies and very few services are being constructed from 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 that a simple game-theoretic 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.

7 References

1 Cao, J., Jarvis, S.A., and Saini, S.: 'ARMS: An agent-based resource management system for Grid computing', *Sci. Program.*, 2002, **10**, (2), pp. 135–148
2 Nitzberg, B., and Schopf, J.: 'Current activities in the scheduling and resource management area of the global Grid forum', *Lect. Notes Comput. Sci.*, 2002, **2537**
3 Schopf, J.M.: 'A general architecture for scheduling on the Grid', Preprint ANL/MCS-P1000-1002, Argonne National Laboratory, 2002
4 Spooner, D.P., Jarvis, S.A., Cao, J., Saini, S., and Nudd, G.R.: 'Local Grid scheduling techniques using performance prediction', *IEE Proc., Comput. Digit. Tech.*, 2003, **150**, (2), pp. 87–96
5 Thomas, N., and Mitrani, I.: 'Routing among different nodes where servers break down without losing jobs', *in* 'Quantitative methods in parallel systems' (Springer-Verlag, 1995), pp. 248–261
6 Mitrani, I., and Wright, P.E.: 'Routing in the presence of breakdowns', *Perform. Eval.*, 1994, **20**, (1–3), pp. 151–164
7 Thomas, N., and Bradley, J.T.: 'Decomposing models of parallel queues'. Proc. 4th Int. Workshop on Queueing Networks with Finite Capacity, 2000
8 Thomas, N.: 'The effect of information latency on performance'. Proc. 19th UK Performance Engineering Workshop, University of Warwick, 2003
9 Martin, S., and Mitrani, I.: 'Optimal scheduling among intermittently unavailable servers', *Int. J. Simul.*, (accepted for publication)
10 Haji, K.D.M., Gourlay, I., and Dew, P.: 'A SNAP-based community resource broker using a three-phase commit protocol: a performance study', *Comput. J.*, (accepted for publication)
11 Fitzgerald, S., Foster, I., Kesselman, C., von Laszewski, G., Smith, W., and Tuecke, S.: 'A directory service for configuring high-performance distributed computations'. Proc. 6th IEEE Symp. on High-Performance Computing, 1997, pp. 365–375
12 Frey, J., Tannenbaum, T., Foster, I., Livny, M., and Tuecke, S.: 'Condor-G: A computation management agent for multi-institutional Grids'. Proc. 10th IEEE Symp. on High-Performance Computing, San Francisco, CA, USA, 2001
13 Buyya, R., Abramson, J., and Giddy, J.: 'Nimrod/G: An architecture for a resource management and scheduling system in a global

computational Grid'. 4th IEEE Conf. on High-Performance Computing in the Asia-Pacific Region, China, 2000
14 Berman, F., and Wolski, R.: 'The AppLeS project: A status report'. Proc. 8th NEC Research Symp., Berlin, Germany, 1997
15 Clark, G., Gilmore, S.T., Hillston, J., and Thomas, N.: 'Experiences with the PEPA workbench modelling tools', *IEE Proc. – Softw.*, 1999, **146**, (1), pp. 11–19
16 Bradley, J.T., Dingle, N.J., Gilmore, S.T., and Knottenbelt, W.J.: 'Derivation of passage-time densities in PEPA models using ipc: the Imperial PEPA Compiler'. Proc. 11th IEEE/ACM Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS), University of Central Florida, Florida, USA, 2003, pp. 344–351
17 Argent-Katwala, A., Bradley, J.T., and Dingle, N.J.: 'Expressing performance requirements using regular expressions to specify stochastic probes over process algebra models'. Proc. 4th Int. Workshop on Software and Performance (WOSP), Redwood City, CA, USA, 2004, pp. 49–58
18 Dingle, N.J., Knottenbelt, W.J., and Harrison, P.G.: 'HYDRA: HYpergraph-based Distributed Response-time Analyser'. Proc. 2003 Int. Conf. on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, USA, 2003, vol. 1, pp. 215–219
19 Woodside, C.M., Neilson, J.E., Petriu, D.C., and Majumdar, S.: 'The stochastic rendezvous network model for performance of synchronous client-server-like distributed software', *IEEE Trans. Comput.*, 1995, **44**, (1), pp. 20–34