Modelling job allocation where service duration is unknown

Nigel Thomas*

University of Newcastle upon Tyne

Abstract

In this paper a novel job allocation scheme in distributed systems is modelled using the Markovian process algebra PEPA. This scheme requires no prior knowledge of job size and has been shown to be more efficient than round robin and random allocation when the job size distribution is heavy tailed and the load is not high. In this paper the job size distribution is assumed to be of a phase-type. Numerical results are derived and compared with those derived from models employing random allocation and the shortest queue strategy.

1 Introduction

In assigning jobs to servers in distributed systems we would like to be able to assign jobs to servers according to the size of the jobs. Thus a large job stuck in a queue behind other large jobs will experience a large delay and a small job in a queue of small jobs will experience a smaller overall delay. If the graduation of scale is set appropriately (i.e. there is sufficient resource to satisfy the demand at each job size) then the proportion of response time versus service time for any size of job should be approximately constant.

This form of scheduling requires some knowledge of the service demand of the jobs. However, in many practical situations this information is unavailable, inaccurate or costly to compute. In such scenarios there are a number of obvious solutions.

- Pull jobs from a central resource. This requires an efficient protocol to manage the movement of jobs and suffers from communication overhead and single point of failure. However, it can provide an efficient mechanism in many situations. This scenario is not considered further in this paper.
- Assign jobs to the service centre with the shortest queue.
- Assign jobs to service centres on a round robin basis.

* *nigel.thomas@ncl.ac.uk*

- Assign jobs to service centres randomly according to probabilities based on service capacity.

The final option is illustrated in Figure 1. The last two options are simple but suffer from the obvious drawback that some jobs may become badly delayed by being stuck behind a large job. If short jobs are so delayed then the proportion of response time versus service time is extremely large.

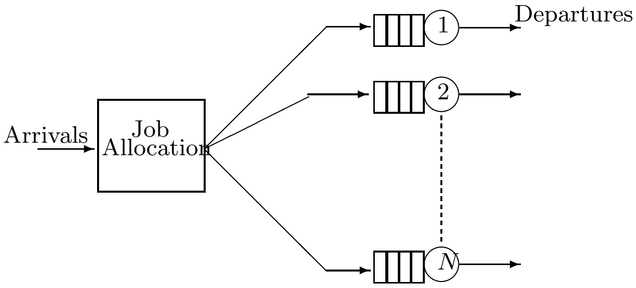

Figure 1: Jobs allocated randomly across N nodes

Harchol-Balter [4] introduced a new algorithm (known as TAG) for allocating jobs where the service demand is not known. In this approach all jobs are sent initially to a single server queue. The server will service the head job either until completion or until some fixed time has expired. If the service is complete then the job departs successfully. However, if the fixed service time was exhausted then the job is passed to the second node. Here a similar process is undertaken, although the fixed timeout is somewhat longer. The process is repeated progressing through nodes with longer and longer timeouts until the final stage where there is no timeout and the job is simply serviced to exhaustion.[1] The system is illustrated in Figure 2.

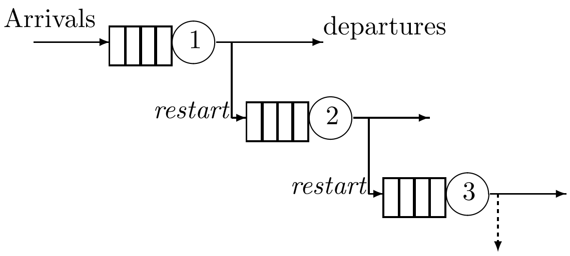

Figure 2: Jobs allocated to nodes using the TAG algorithm

The efficiency of this system is highly dependent on appropriately setting

[1]This scheme differs from traditional multi-level feedback queueing in that here jobs are killed if they reach the end of the timeout at a node and are then restarted at the next node. In multi-level feedback queueing the job would be transferred to the next node and service resumed, therefore no effort would be lost.

the timeouts. These must be set so that some jobs complete successfully at each node and some timeout. Even so the scheme is somewhat counter intuitive. Service capacity is apparently wasted by trying to serve jobs that ultimately time out. However, this mechanism ensures that jobs which can complete within this timeout will not be delayed for a long time by much larger jobs. In addition a larger job will experience many periods of repeated service, perhaps ultimately receiving many times its service demand before it completes. However, this repeated service affects jobs according to their size, thus a large job will experience more repeated services than a smaller job. Hence, it should be clear that small jobs should progress relatively quickly and large jobs will experience much longer delays. In addition, for all but the largest jobs the delay is bounded.

The advantage of using this scheme is highly dependent on the distribution of the service demands. Take the following simple illustration. Suppose we have six jobs awaiting service, each node has a service rate of 1, and the service demands of the jobs are as follows, $\{4, 5, 6, 7, 3, 2\}$, in seconds. If there is no timeout set (or the timeout is greater than 7 seconds in this case) then all the jobs would be served in order at the first node, and the average response time would be 17 seconds. Similarly if the timeout was zero, all the jobs would be served at the second node and the average response time would be the same. If the timeout is increased to 1.5 seconds, then no advantage is gained because all jobs will still timeout at the first server, the average response time being 18.5 seconds. If the timeout is further increased to 3.5 seconds then the final two jobs will complete at the first server, but only after the preceding four have timed out and proceeded to the second node. In this case there is a slight improvement, as the average response time is 16.67 seconds. Further increasing the timeout allows more jobs to successfully complete at the first node, however in this case the minimum response time of 15.67 seconds would attained with a timeout fractionally above 3 seconds. If the service demands of the jobs were, $\{99, 5, 6, 7, 3, 2\}$, in seconds, then a much more dramatic gain can be made. Here the optimal timeout is (predictably) fractionally above 7 seconds, where the average response time is 36.5 seconds, as opposed to the no timeout case of 112 seconds.

Harchol-Balter [4] showed that job streams with a heavy tailed distribution of service demand will benefit significantly from this scheme according to a metric called *mean slowdown*. The aim of this paper is to model the TAG algorithm using the Markovian process algebra PEPA and to investigate the effectiveness of TAG when the queues are finite.

The PEPA model of the TAG algorithm is defined in Section 2, followed by an alternative model and extensions using phase type distributions. Section 3 presents some numerical results, which is followed by a discussion of some related work and future oportunities.

2 The Model

Because PEPA is Markovian it is necessary to model the timeouts as an Erlang distribution rather than deterministically and the service distribution is negative exponential (although certain phase type distributions are also possible, see Section 2.2).

$$
\begin{aligned}
Queue1_0 &\stackrel{def}{=} (arrival, a).Queue1_1 \\
Queue1_i &\stackrel{def}{=} (arrival, a).Queue1_{i+1} + (service1, \top).Queue1_{i-1} \\
&\quad +(timeout, \top).Queue1_{i-1} + (tick1, \top).Queue1_i \ , 1 \leq i < K_1 \\
Queue1_{K_1} &\stackrel{def}{=} (timeout, \top).Queue1_{K_1-1} + (service1, s).Queue1_{K_1-1} \\
&\quad +(tick1, \top).Queue1_{K_1}
\end{aligned}
$$

$$
Server1 \stackrel{def}{=} (service1, s).Server1
$$

$$
\begin{aligned}
Time1_0 &\stackrel{def}{=} (timeout, t).Time1_n + (service1, \top).Time1_n \\
Time1_i &\stackrel{def}{=} (tick1, t).Time1_{i-1} + (service1, \top).Time1_n \ , 1 \leq i \leq n
\end{aligned}
$$

$$
\begin{aligned}
Queue2_0 &\stackrel{def}{=} (timeout, \top).Queue2_1 \\
Queue2_i &\stackrel{def}{=} (timeout, \top).Queue2_{i+1} + (repeatservice, \top).Q2_i \\
&\quad +(tick2, \top).Queue2_i \ , 1 \leq i < K_2 \\
Q2_i &\stackrel{def}{=} (timeout, \top).Q2_{i+1} + (service2, s).Queue2_{i-1} \\
&\quad +(tick2, \top).Q2_i \ , 1 \leq i < K_2 \\
Queue2_{K_2} &\stackrel{def}{=} (timeout, \top).Queue2_{K_2} + (repeatservice, \top).Q2_{K_2} \\
&\quad +(tick2, \top).Queue2_{K_2} \\
Q2_{K_2} &\stackrel{def}{=} (timeout, \top).Q2_{K_2} + (service2, s).Queue2_{K_2-1} \\
&\quad +(tick2, \top).Q2_{K_2}
\end{aligned}
$$

$$
Server2 \stackrel{def}{=} (service2, s).Server2
$$

$$
\begin{aligned}
Time2_0 &\stackrel{def}{=} (repeatservice, t).(service2, \top).Time2_n \\
Time2_i &\stackrel{def}{=} (tick2, t).Time2_{i-1} \ , 1 \leq i \leq n
\end{aligned}
$$

$$
\begin{aligned}
Node_1 &\stackrel{def}{=} Server1 \underset{\{service1\}}{\bowtie} Queue1_0 \underset{\left\{\substack{timeout,\\ service1, tick1}\right\}}{\bowtie} Time1_n \\
Node_2 &\stackrel{def}{=} Server2 \underset{\{service2\}}{\bowtie} Queue2_0 \underset{\left\{\substack{repeatservice,\\ service2, tick2}\right\}}{\bowtie} Time2_n
\end{aligned}
$$

$$Node_1 \underset{\{timeout\}}{\bowtie} Node_2$$

Figure 3: A PEPA model of a two node system employing TAG

This model uses a few common mechanisms that are familiar from other queueing papers (see [9] for example). The deterministic timeout at the first node is modelled as an Erlang process with arbitrary number of ticks (as is conventional). Prior to service, the timeout clock must be started, this is done by enabling the *tick* actions in every derivative of the queues except when the queue is empty. The timeout is raced against a service process *service1*; note that this is negative exponentially distributed, but could be replaced by any feasible phase type distribution subject to the consequential state space implications and calculation of the residual service to be completed at the second node (see Section 2.2). If *service1* wins the race then the job departs, otherwise it proceeds to $Node_2$. The timeout is reset and the race begins again with a new job if one is in the queue, or is idle until next arrival if the queue is empty.[2]

At the second node the amount of service from the timeout period must be repeated, this is included in the action *repeatservice*. In the exponential case the remaining service has the same distribution as the original service distribution, hence at node 2 the job receives a repeat service for the part that was already performed at node 1, and then a service for the remaining part. It is assumed that the nodes are identical, hence the service rates are the same. However, if the system is heterogeneous, then it would be necessary to introduce new rates for the ticks of the repeated service and for *service2*. It would be a simple matter to add more nodes to the model in the same fashion.

2.1 An alternative model

The model illustrated in Figure 3 gives rise to a CTMC with $(K_1(n+1)+1)(K_2(n+2)+1)$ states. An alternative representation of the queue components is possible whereby each place in the queue is modelled separately and the whole queue is a parallel composition of all the places. Thus, the queues shown in Figure 3 would be modelled as follows.

$$\begin{aligned}
Q1_0 &\stackrel{def}{=} (arrival, \top).Q1_1 \\
Q1_1 &\stackrel{def}{=} (timeout, \top).Q1_0 + (service1, \top).Q1_0 + (tick1, \top).Q1_1
\end{aligned}$$

$$Q2_0 \stackrel{def}{=} (timeout, \top).Q2_1$$

[2] It would appear to be sensible to continue serving this job until it completes or an arrival occurs, if the queue is otherwise empty, however this is not part of the TAG algorithm. Such a mechanism can be modelled by removing the *timeout* action from $Queue1_1$ and introducing a separate arrival action in $Queue1_1$ as an immediate trigger.

$$Q2_1 \stackrel{def}{=} (repeatservice, \top).(service2, \top).Q2_0 + (tick2, \top).Q2_1$$

$$Arrivals \stackrel{def}{=} (arrival, a).Arrivals$$

$$Queue1 \stackrel{def}{=} Arrivals \underset{\{arrival\}}{\bowtie} (Q1_0|\ldots|Q1_0)$$

$$Node_1 \stackrel{def}{=} Time1_n \underset{\left\{\substack{timeout,\\ service1, tick1}\right\}}{\bowtie} Queue1 \underset{\{service1\}}{\bowtie} Server1$$

$$Node_2 \stackrel{def}{=} Time2_n \underset{\left\{\substack{repeatservice,\\ service2, tick2}\right\}}{\bowtie} (Q2_0|\ldots|Q2_0) \underset{\{service2\}}{\bowtie} Server2$$

Figure 4: An alternative PEPA model of a two node system employing TAG

Traditionally this kind of representation would not be used because there is a much larger state space in the underlying CTMC than for the state based representation given in Figure 3. This is because there are now many states representing each number of jobs in the queues. However, this style of model is amenable to a form of analysis based on ordinary differential equations that has recently been applied to PEPA [7] and is supported by the Dizzy tool [8]. This analysis effectively counts the number of components in a given derivative without recourse to deriving the underlying CTMC. Thus, it is possible to count the number of components behaving as derivative $Q1_0$ to calculate the number of jobs in the first queue. This form of analysis is extremely efficient and so even though the underlying CTMC is much larger, it is still possible to analyse models with far more derivatives (in this case larger queues).

It is important to note that in both cases the maximum queue length is bounded. This means jobs can be lost from the system, either by being dropped at node 1 on arrival, or at node 2 after completing a timed-out service at node 1. This latter case is particularly important as it means that work is being undertaken on jobs which are ultimately unsuccessful. Ultimately this means that if the load is sufficiently high and the timeout is set too short then a significant proportion of jobs may be dropped at the second node. Thus, it is important to look at throughput and average response time as the main measures of this finite system. The average response time can be calculated by Little's Law from the average queue length and the average arrival rate of successful jobs.

2.2 Phase-type distributions

The exponential distribution is not the most interesting to employ when considering the TAG algorithm. One reason for this is we know that the optimal task assignment policy for exponential arrivals and services is the shortest queue policy (sending the job to the node with the least number of jobs). Thus TAG is always going to be suboptimal. TAG becomes more useful when the variability of the service demand distribution increases. Although PEPA cannot be used to model general distributions, it can be used to specify phase type distributions.

Phase type distributions are distributions constructed by combining multiple exponential random variables. The Erlang distribution, used in Figure 3, is an example of a phase type distribution which consists of an exponential distribution repeated k times. Another important phase type distribution is the hyper-exponential, or H_k, distribution, which is a random choice between k exponential distributions. The most commonly used hyper-exponential is the H_2-distribution, which has three parameters, α, λ_1 and λ_2 and the following cumulative distribution function.

$$F_{H_2} = 1 - \alpha e^{-\lambda_1 t} - (1-\alpha)e^{-\lambda_2 t} \ , \ t \geq 0.$$

The residual life of an H_2 random variable following an Erlang is easily calculated. As might be expected the result has an H_2-distribution, although with parameters α', λ_1 and λ_2. An important feature of the hyper-exponential distribution is that it has a greater variance than an exponential distribution of the same mean (as long as $\lambda_1 \neq \lambda_2$ obviously). This is in contrast to the Erlang distribution, where the variance decreases as k increases, so that for large k the Erlang distribution is approximately deterministic. As such the hyper-exponential would be ideal for using as a service distribution with the TAG algorithm.

Unfortunately, it is not possible to specify the H_2 distribution exactly in PEPA, as PEPA lacks explicit probabilistic branching required to start the distribution. This can be approximated as follows:

$$\begin{aligned} StartH_2 \stackrel{def}{=} & \ (branch, \alpha \times fast).(phase1, \lambda_1).End \\ & +(branch, (1-\alpha) \times fast).(phase2, \lambda_2).End \end{aligned}$$

where *fast* is a large number, so that the *branch* action does not distort the distribution.

It is possible to exactly specify another class of phase type distributions, the *Cox distribution*. The Cox distribution is formed from a sequence of exponentials, rather like the Erlang, although following each phase there is a probabilistic choice as to whether to proceed to the next phase or terminate. The coefficient of variation is dependent on the choice of parameters, and so the Cox distribution can be used to approximate almost deterministic processes, through to those with a very high variance. A very simple Cox distribution can be modelled in PEPA as follows.

$$StartC_2 \stackrel{def}{=} (phase1, \alpha\lambda_1).End + (phase1, (1-\alpha)\lambda_1).(phase2, \lambda_2).End$$

If $\lambda_2 >> \lambda_1$ and $1 > \alpha \geq 0.5$ then this component will give rise to a delay between *Start* and *End* with a variance greater than that for an exponential distribution with the same mean.

The calculation of the residual life of this two phase Cox distribution following the Erlang timeout results in a phase type distribution of a slightly different form, which can be expressed approximately in PEPA as follows:

$$\begin{aligned} StartR &\stackrel{def}{=} (branch, p \times fast).(phase2, p\lambda_2).End \\ &\quad +(branch, (1-p) \times fast).StartC_2 \end{aligned}$$

where *fast* is a suitably large number and p is given by the result of the race between *phase1* and the Erlang distributed timeout, conditional on the Erlang distribution winning the race against the Cox distribtution.

Incorporating a Cox_2 distributed service demand into the model in Figure 3 is a matter of redefining the *Server1* and *Server2* components.

$$\begin{aligned} Server1 &\stackrel{def}{=} (service1, \alpha s_1).Server1 + (phase1, (1-\alpha)s_1).Server1' \\ &\quad +(timeout, \top).Server1 \\ Server1' &\stackrel{def}{=} (service1, s_2).Server1 + (timeout, \top).Server1 \\ Server2 &\stackrel{def}{=} (branch, p \times fast).(service2, s_2).Server2 \\ &\quad +(branch, (1-p) \times fast).Server2' \\ Server2' &\stackrel{def}{=} (service1, \alpha s_1).Server2 \\ &\quad +(phase1, (1-\alpha)s_1).(service2, s_2).Server2 \end{aligned}$$

It is also necessary to add additional terms of $+(phase1, \top).Queue1_i$ to every $Queue1_i$ where $i \geq 1$, and $+(phase2, \top).Q2_i$ to every $Q2_i$ where $i \geq 1$. It is also necessary to add $phase1$ and *timeout* to the cooperation set between $Server1$ and $Queue1_0$, and *branch* to the cooperation set between $Server2$ and $Queue2_0$. These changes ensure that the $phase1$ and *branch* actions are blocked when the queues are empty and that the timeout resets the service process. The resulting model has $(2(n+1)K_1 + 1)((n+5)K_2 + 1)$ states.

3 Numerical results

The model specified in Figure 3 is analysed with $n = 6$ and $K_1 = K_2 = 10$. This gives rise to a model of 4331 states. Figure 5 shows the effect of the timeout rate on the average queue size, plots are also included for random job assignment and the (optimal) shortest queue assignment strategies by way of contrast.

Figure 6 shows the average response times for the same systems. It is important to note that whereas the shortest queue strategy has almost negligible loss at this arrival rate, the random assignment and TAG are somewhat higher, although still less than 10^{-4}. Clearly, as the job loss rate is so low, there is little difference between the curves for TAG in Figures 5 and 6.

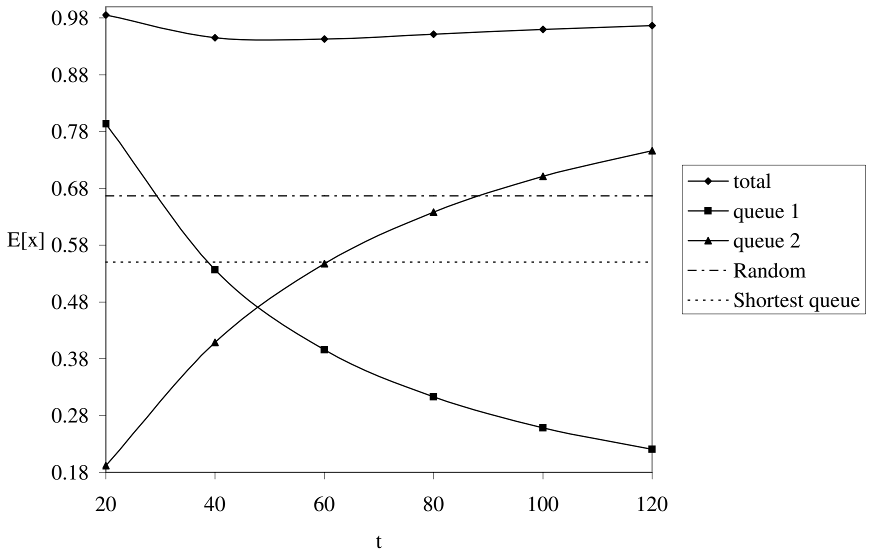

Figure 5: Average queue length varied against timeout rate, $a = 5$, $s = 10$

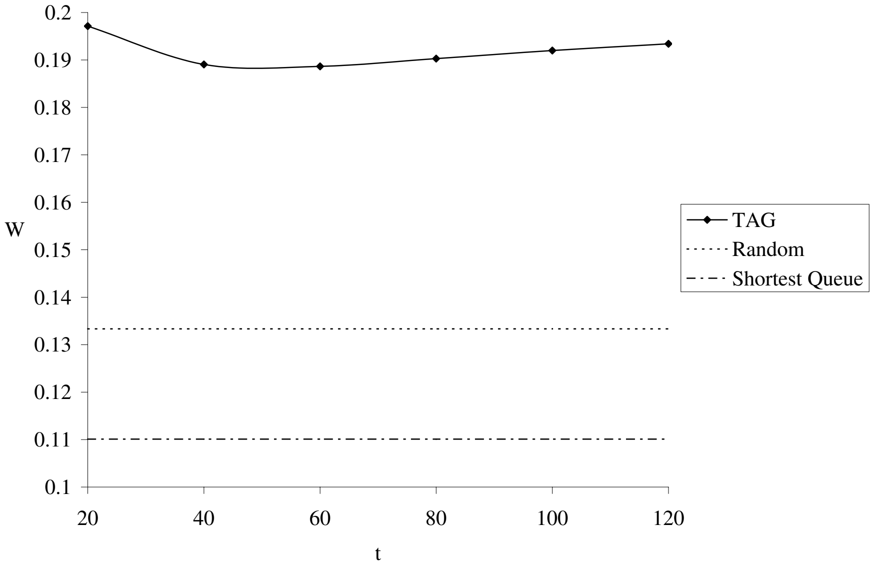

Figure 6: Average response time varied against timeout rate, $a = 5$, $s = 10$

In Figure 7 the same system is shown with varying arrival rates. The TAG algorithm is optimised for minimum queue length, the optimal (integer) values of t being 42, 45, 49 and 51 (for a =11, 9, 7 and 5 respectively). Again these are compared with the average response times for random allocation and the shortest queue strategy.

It would appear from these results that TAG isn't very good compared with

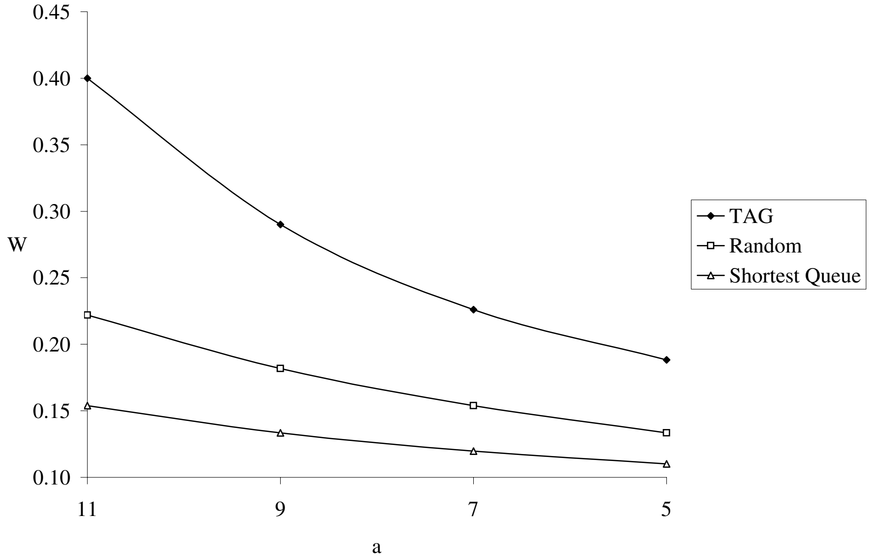

Figure 7: Average response time varied against arrival rate, $s = 10$

the random and shortest queue strategies. This is particularly the case as the load increases, when the extra, incomplete, service in TAG has a greater effect. This shouldn't be a great surprise as it is well known that the shortest queue strategy is the optimal task assignment strategy for exponential arrivals and service demands. However, for a service demand with greater variance, TAG would be expected to perform better.

Because of its unusual structure, TAG will give different optimal values of t according to what metric is being considered. In the course of this experimentation it has been noted that utilisation, average response time and throughput, are all maximised or minimised at slightly different values of t. This is also recognised in [4], where different optimisations are presented for slowdown, waiting time and fairness.[3] However, the results derived here differ from those in [4], where the load on the different queues was deliberately unbalanced. This remains an area of future experimentation.

4 Related Work

Models of queues modelled with stochastic process algebra have appeared in a number of papers, amongst the more in depth are those by Bernardo et al [1], Herzog and Mertsiotakis [5] and Thomas and Hillston [9]. Thomas and Hillston [9] covered a number of different queueing scenarios and showed how PEPA could be used to model these. The model specified in Figure 3 is based on that earlier approach.

[3]The optimisation for fairness seeks to make the slowdown nearly constant regardless of job length.

The analysis of job allocation algorithms has a long and colourful history. The approach used here is based on a paper by Mor Harchol-Balter [4]. Related earlier studies were conducted by Crovella et al [3] and Bestavros [2]. In each of these cases it was assumed that a job could not be interrupted and service resumed from the same point, but rather that any stopped job must be entirely restarted. All three of these papers concern unbounded queues, whereas this paper is concerned with bounded (finite) queues. There are a significant number of studies covering the related case where a job can be migrated, or otherwise resumed from point of interruption. In such instances the optimal policy is generally quite different, depending on the cost of migration.

As yet nobody as studied the costs and benefits of resume against restart following job transfer. As such this remains an interesting open problem. This topic also bears some relation to a problem of optimal interrupt and restart times studied by van Moorsel and Wolter [10].

5 Conclusions and further work

A model of a novel allocation strategy has been presented in this paper using PEPA. The model is quite complex and illustrates how PEPA can be used to formally model systems which are not intuitively process oriented or obviously Markovian. This paper therefore extends the class of queueing systems that have been modelled using PEPA.

It has been necessary to introduce some approximations in the model, most significantly Erlang distributions representing deterministic delays, as well as a branching process to approximate the residual life of a Cox distribution. The degree of error introduced by these approximations has not been investigated in this paper, but is left for future work. A simulation of the allocation scheme is being developed that will be used for this purpose.

It has been assumed that the job services are exponentially distributed. It would also be possible to model phase type distributions, and an approach to using Cox distributions is described. In the simulator it will be possible to use any service distribution. Therefore future work will include comparing the results from the simulator with the PEPA model using phase type approximations.

Acknowledgements

The author would like to thank Dr Uli Harder of Imperial College London for introducing him to this area. A simulator is currently under development by Alex Liu as part of his MSc project at the University of Newcastle upon Tyne.

References

[1] M. Bernardo, L. Donatiello and R. Gorrieri, Describing Queueing Systems with MPA, Technical Report UBLCS-94-11, University of Bologna, May,

1994.

[2] A. Bestavros, Load profiling: A methodology for scheduling realtime tasks in a distributed system. in: *Proceedings of ICDCS '97*, May 1997.

[3] M. Crovella, M. Harchol-Balter, and C. Murta, Task assignment in a distributed system: Improving performance by unbalancing load, in: *Proceedings of ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems Poster Session*, 1998.

[4] M. Harchol-Balter, Task Assignment with Unknown Duration, Journal of the ACM , **49**(2), pp. 260-288. 2002.

[5] U. Herzog and V. Mertsiotakis, Stochastic Process Algebras Applied to Failure Modelling, in U. Herzog and M. Rettelbach eds., *Proceedings of the 2nd Workshop on Process Algebra and Performance Modelling*, Regensberg, Germany, July, 1994.

[6] J. Hillston, *A Compositional Approach to Performance Modelling*, vol. 12 of *Distinguished Dissertations in Computer Science*, Cambridge University Press, 1996.

[7] J. Hillston, Fluid Flow Approximation of PEPA models,in *Proceedings of the 2nd International Conference on the Quantitative Evaluation of Systems (QEST)*, 2005.

[8] S. Ramsey, D. Orrell, and H. Bolouri, Dizzy: stochastic simulation of large-scale genetic regulatory networks, *Journal of Bioinformatics and Computational Biology*, **3**(2), pp. 415-436, 2005.

[9] N. Thomas and J. Hillston, Using Markovian Process Algebra to Specify Interactions in Queueing Systems, Technical Report LFCS-97-373, Department of Computer Science, University of Edinburgh, 1997.

[10] A. van Moorsel and K. Wolter, Optimal Restart Times for Moments of Completion Time, *IEE Proceedings - Software*, **151**(5), pp 219-223, 2004