Modelling job allocation where service duration is unknown

Nigel Thomas
School of Computing Science
University of Newcastle
UK
nigel.thomas@ncl.ac.uk

Abstract

In this paper a novel job allocation scheme in distributed systems (TAG) 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 and the queues are bounded. Numerical results are derived and compared with those derived from models employing random allocation and the shortest queue strategy. It is shown that TAG can perform well for a range of performance metrics.

1 Introduction

Ideally in assigning jobs to servers in performance oriented distributed systems we would like to be able to assign jobs to servers according to the size of the jobs. This enables the optimisation of the system according to some performance criterion. Conventionally this may entail minimising the average response time or optimising the response time of a job according to its service demand. Thus a large job stuck in a queue behind other large jobs may experience a relatively large delay and a small job in a queue of small jobs may 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 can suffer from communication latency 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.
- 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.

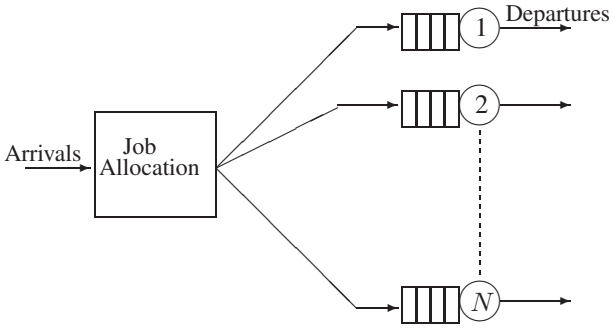

Figure 1. Jobs allocated randomly across N nodes

Harchol-Balter [5] 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

1-4244-0054-6/06/$20.00

simply serviced to exhaustion. The system is illustrated in Figure 2.

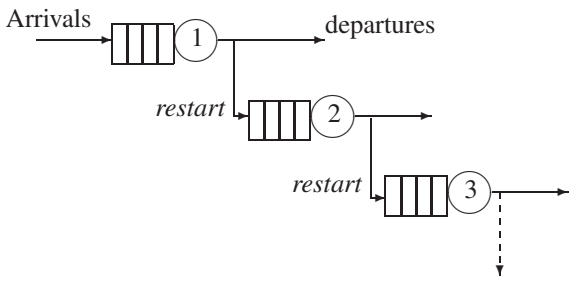

Figure 2. Jobs allocated nodes using the TAG algorithm

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 efficiency of this system is highly dependent on appropriately setting 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 [5] 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. It is assumed that all jobs take an equal amount of space in the queue, therefore it is necessary only to count the number of jobs, and not the amount of buffer space used. It may appear to be a significant restriction, however there are many applications where the job size is nearly constant and yet there is significant variance in service duration.

- Simulation; where simply changing the simulation time may alter the service duration by many orders of magnitude. This is particularly true when considering optimisation, when crude approximation may be used to roughly find optimal parameters, but increasing accuracy is needed as the optimal values are refined.
- Visualisation; where the resolution will greatly affect rendering time, but the data set remains the same.
- Database queries; while these may vary in size, the size of a query will generally be small enough to be treated atomically and there is no clear relation between the size of the query and the time taken to return its result.

In this paper the queues are bounded, jobs can therefore 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.

The following section briefly introduces PEPA. This is followed by a PEPA model of the TAG algorithm, an alternative model, extensions using phase type distributions and some simple approximations. Section 5 presents some numerical results, which is followed by a discussion of some related work and future opportunities.

2 PEPA

A formal presentation of PEPA is given in [7], in this section a brief informal summary is presented. PEPA, being a Markovian Process Algebra, only supports actions that occur with rates that are negative exponentially distributed. Specifications written in PEPA represent Markov processes and can be mapped to a continuous time Markov chain (CTMC). Systems are specified in PEPA in terms of *activities* and *components*. An activity (α, r) is described by the *type* of the activity, α, and the *rate* of the associated negative exponential distribution, r. This rate may be any positive real number, or given as *unspecified* using the symbol $\top$. The syntax for describing components is given as:

$$P ::= (\alpha, r).P \mid P + Q \mid P/L \mid P \underset{L}{\bowtie} Q \mid A$$

The component $(\alpha, r).P$ performs the activity of type α at rate r and then behaves as P. The component $P + Q$ behaves either as P or as Q, the resultant behaviour being given by the first activity to complete.

The component P/L behaves exactly like P except that the activities in the set L are concealed, their type is not visible and instead appears as the unknown type τ.

Concurrent components can be synchronised, $P \underset{L}{\bowtie} Q$, such that activities in the *cooperation set* L involve the participation of both components. In PEPA the shared activity occurs at the slowest of the rates of the participants and if a rate is unspecified in a component, the component is passive with respect to activities of that type. The parallel combinator $\|$ is used as shorthand to denote synchronisation with no shared activities, i.e. $P \| Q \equiv P \underset{\emptyset}{\bowtie} Q$. $A \stackrel{def}{=} P$ gives the constant A the behaviour of the component P.

In this paper we consider only models which are cyclic, that is, every derivative of components P and Q are reachable in the model description $P \underset{L}{\bowtie} Q$. Necessary conditions for a cyclic model may be defined on the component and model definitions without recourse to the entire state space of the model.

3 The Model

The TAG system is now modelled in PEPA. Since PEPA is Markovian, it is necessary to model the TAG timeouts as an Erlang distribution rather than deterministically. The service distribution is initially assumed to be negative exponential, although certain phase type distributions are also possible (see Section 3.2). Queues are modelled in a state based fashion, depicting each number of jobs in a queue as a separate named derivative of the queue component. An alternative specification of the queues is given in Section 3.1.

Figure 3 shows a model of the TAG scheme for a two node system.

$$
\begin{aligned}
Q1_0 &\stackrel{def}{=} (arrival, \lambda).Q1_1 \\
Q1_i &\stackrel{def}{=} (arrival, \lambda).Q1_{i+1} + (service1, \mu).Q1_{i-1} \\
&\quad +(timeout, \top).Q1_{i-1} + (tick1, \top).Q1_i \\
&\quad , \ 1 \leq i < K_1 \\
Q1_{K_1} &\stackrel{def}{=} (timeout, \top).Q1_{K_1-1} + (tick1, \top).Q1_{K_1} \\
&\quad +(service1, \mu).Q1_{K_1-1}
\end{aligned}
$$

$$
\begin{aligned}
Timer1_0 &\stackrel{def}{=} (timeout, t).Timer1_n \\
&\quad +(service1, \top).Timer1_n \\
Timer1_i &\stackrel{def}{=} (tick1, t).Timer1_{i-1} \\
&\quad +(service1, \top).Timer1_n \ , \ 1 \leq i \leq n
\end{aligned}
$$

$$
\begin{aligned}
Q2_0 &\stackrel{def}{=} (timeout, \top).Q2_1 \\
Q2_i &\stackrel{def}{=} (timeout, \top).Q2_{i+1} + (tick2, \top).Q2_i \\
&\quad +(repeatservice, \top).Q2'_i \ , \ 1 \leq i < K_2 \\
Q2'_i &\stackrel{def}{=} (timeout, \top).Q2'_{i+1} + (tick2, \top).Q2'_i \\
&\quad +(service2, \mu).Q2_{i-1} \ , \ 1 \leq i < K_2 \\
Q2_{K_2} &\stackrel{def}{=} (timeout, \top).Q2_{K_2} + (tick2, \top).Q2_{K_2} \\
&\quad +(repeatservice, \top).Q2'_{K_2} \\
Q2'_{K_2} &\stackrel{def}{=} (timeout, \top).Q2'_{K_2} + (tick2, \top).Q2'_{K_2} \\
&\quad +(service2, \mu).Q2_{K_2-1}
\end{aligned}
$$

$$
\begin{aligned}
Timer2_0 &\stackrel{def}{=} (repeatservice, t).Timer2_n \\
Timer2_i &\stackrel{def}{=} (tick2, t).Timer2_{i-1} \ , \ 1 \leq i \leq n
\end{aligned}
$$

$$
\begin{aligned}
Node_1 &\stackrel{def}{=} Q1_0 \underset{\left\{\substack{timeout, \\ service1, tick1}\right\}}{\bowtie} Timer1_n \\
Node_2 &\stackrel{def}{=} Q2_0 \underset{\left\{\substack{repeatservice, \\ service2, tick2}\right\}}{\bowtie} Timer2_n
\end{aligned}
$$

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

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 [10] 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. 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.

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 is a simple matter to add more nodes to the model in the same fashion.

3.1 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 shown in Figure 4.

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 potentially amenable to a form of analysis based on ordinary differential equations that has recently been applied to PEPA [8] and is supported by the Dizzy tool [9]. 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).

3.2 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

$$
\begin{aligned}
Q1_0 &\stackrel{def}{=} (arrival, \top).Q1_1 \\
Q1_1 &\stackrel{def}{=} (timeout, \top).Q1_0 + (service1, \top).Q1_0 \\
&\quad +(tick1, \top).Q1_1 \\
Q2_0 &\stackrel{def}{=} (timeout, \top).Q2_1 \\
Q2_1 &\stackrel{def}{=} (repeatservice, \top).(service2, \top).Q2_0 \\
&\quad +(tick2, \top).Q2_1 \\
S1 &\stackrel{def}{=} (arrival, \lambda).S1 + (service1, \mu).S1 \\
S2 &\stackrel{def}{=} (service2, \mu).S2
\end{aligned}
$$

$$
Queue1 \stackrel{def}{=} S1 \underset{\left\{\substack{arrival \\ service1}\right\}}{\bowtie} (Q1_0 | \ldots | Q1_0)
$$

$$
Queue2 \stackrel{def}{=} S2 \underset{\{service2\}}{\bowtie} (Q2_0 | \ldots | Q2_0)
$$

$$
Node_1 \stackrel{def}{=} Timer1_n \underset{\left\{\substack{timeout, \\ service1, tick1}\right\}}{\bowtie} Queue1
$$

$$
Node_2 \stackrel{def}{=} Timer2_n \underset{\left\{\substack{repeatservice, \\ service2, tick2}\right\}}{\bowtie} Queue2
$$

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

general distributions, it can be used to specify phase type distributions.

Phase type distributions are distributions constructed by combining multiple exponential random variables. These can be used to approximate most general distributions and approximations can be constructed using tools such as *EMpht* [1]. 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, μ_1 and μ_2 and the following cumulative distribution function.

$$F_{H_2} = 1 - \alpha e^{-\mu_1 t} - (1-\alpha)e^{-\mu_2 t}\ ,\ t \geq 0.$$

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 $\mu_1 \neq \mu_2$ 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.

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', μ_1 and

μ_2. Modelling the H_2 distribution in PEPA requires the introduction of several additional terms in order to generate the necessary probabilistic branching. Thus each $service1$ and $timeout$ action must occur twice, with the rates multiplied by α and $1-\alpha$, in order to determine whether the next job will be served at rate μ_1 or μ_2 (in derivatives $Q1_i$ and $Q1'_i$ respectively). The exception is in $Q1_1$, when these actions result in the queue becoming empty. Hence an $arrival$ action in $Q1_0$ will perform the necessary branching instead. In the second node the situation is simpler, with the branching occurring at the $repeatservice$ action.

Hence the following modifications are made to the TAG model (all other components remain as previously specified).

$$
\begin{aligned}
Q1_0 &\stackrel{def}{=} (arrival, \alpha\lambda).Q1_1 + (arrival, (1-\alpha)\lambda).Q1'_1 \\
Q1_1 &\stackrel{def}{=} (arrival, \lambda).Q1_2 + (tick1, \top).Q1_1 \\
&\quad +(service1, \mu_1).Q1_0 + (timeout, t).Q1_0 \\
Q1_i &\stackrel{def}{=} (arrival, \lambda).Q1_{i+1} + (tick1, \top).Q1_i \\
&\quad +(service1, (1-\alpha)\mu_1).Q1'_{i-1} \\
&\quad +(service1, \alpha\mu_1).Q1_{i-1} \\
&\quad +(timeout, (1-\alpha)\mu_2).Q1'_{i-1} \\
&\quad +(timeout, \alpha\mu_2).Q1_{i-1} \ , \ 2 \le i < K_1 \\
Q1_{K_1} &\stackrel{def}{=} (tick1, \top).Q1_{K_1} + (timeout, \alpha t).Q1_{K_1-1} \\
&\quad +(timeout, (1-\alpha)t).Q1'_{K_1-1} \\
&\quad +(service1, (1-\alpha)\mu_1).Q1'_{K_1-1} \\
&\quad +(service1, \alpha\mu_1).Q1_{K_1-1} \\
Q1'_1 &\stackrel{def}{=} (arrival, \lambda).Q1'_2 + (tick1, \top).Q1'_1 \\
&\quad +(service1, \mu_2).Q1_0 + (timeout, t).Q1_0 \\
Q1'_i &\stackrel{def}{=} (arrival, \lambda).Q1'_{i+1} + (tick1, \top).Q1'_i \\
&\quad +(service1, (1-\alpha)\mu_2).Q1'_{i-1} \\
&\quad +(service1, \alpha\mu_2).Q1_{i-1} \\
&\quad +(timeout, (1-\alpha)t).Q1'_{i-1} \\
&\quad +(timeout, \alpha t).Q1_{i-1} \ , \ 2 \le i < K_1 \\
Q1'_{K_1} &\stackrel{def}{=} (tick1, \top).Q1'_{K_1} + (timeout, \alpha t).Q1_{K_1-1} \\
&\quad +(timeout, (1-\alpha)t).Q1'_{K_1-1} \\
&\quad +(service1, (1-\alpha)\mu_2).Q1'_{K_1-1} \\
&\quad +(service1, \alpha\mu_2).Q1_{K_1-1}
\end{aligned}
$$

$$
\begin{aligned}
Q2_i &\stackrel{def}{=} (timeout, \top).Q2_{i+1} + (tick2, \top).Q2_i \\
&\quad +(repeatservice, \alpha' t).Q2'_i \\
&\quad +(repeatservice, (1-\alpha')t).Q2''_i \\
&\quad , \ 1 \le i < K_2 \\
Q2'_i &\stackrel{def}{=} (timeout, \top).Q2'_{i+1} + (service2, \mu_1).Q2_{i-1} \\
&\quad , \ 1 \le i < K_2 \\
Q2''_i &\stackrel{def}{=} (timeout, \top).Q2''_{i+1} + (service2, \mu_2).Q2_{i-1} \\
&\quad , \le i < K_2 \\
Q2_{K_2} &\stackrel{def}{=} (timeout, \top).Q2_{K_2} + (tick2, \top).Q2_{K_2} \\
&\quad +(repeatservice, \alpha' t).Q2'_{K_2} \\
&\quad +(repeatservice, (1-\alpha')t).Q2''_{K_2} \\
Q2'_{K_2} &\stackrel{def}{=} (timeout, \top).Q2'_{K_2} + (service2, \mu_1).Q2_{K_2-1} \\
Q2''_{K_2} &\stackrel{def}{=} (timeout, \top).Q2''_{K_2} + (service2, \mu_2).Q2_{K_2-1}
\end{aligned}
$$

Figure 5. Adding hyper-exponential services to the TAG model

4 Simple approximations

A key factor in deploying TAG is obviously the calculation of appropriate timeout values. Failure to optimise these values can lead to a considerable loss of performance. Although an analytic solution of the full model (and hence an exact calculation of the optimal timeouts) is possible, such a calculation is complex and beyond the scope of this paper. Instead a series of approximations are presented here which can be used to obtain estimates of good timeout values.

It is reasonable to assume that a good estimate of the timeout optimised to minimise response time will be found when the average service demand at all nodes is the same in a homogeneous system. Consider therefore, in the first instance, a two node system with unbounded queues and exponential service demands. Those jobs which time out at node 1 will proceed to node 2 where they will receive a repeat service followed by the residual service. On average (in our model), the repeat service at node 2 and the time out period at node 1 will be the same for these jobs. Given that our aim is to balance the service demand, we can therefore restrict the solution to just considering the successfully completing services at node 1 and the residual services at node 2.

If we assume timeout period is negative exponential with mean $1/T$ then the service demands will balance subject to the following condition,

$$\frac{T}{(T+\mu)}\frac{1}{\mu} = \frac{\mu}{(T+\mu)}\frac{1}{(T+\mu)}$$

Hence,

$$\mu^2 = T^2 + T\mu$$

If $\mu = 10$ then this approximation would predict a timeout duration of approximately 6.17. In fact the successfully completing services at node 1 are the result of a race between an exponential service and an Erlang timeout. Thus,

$$
\begin{aligned}
\left(\frac{t}{(t+\mu)}\right)^n \frac{1}{\mu} &= \frac{\mu}{(t+\mu)}\frac{1}{(t+\mu)} + \\
&\quad \frac{\mu}{(t+\mu)}\frac{t}{(t+\mu)}\frac{2}{(t+\mu)} + \\
&\quad \ldots + \\
&\quad \frac{\mu}{(t+\mu)}\left(\frac{t}{t+\mu}\right)^{n-1}\frac{n}{(t+\mu)} \\
&= \frac{\mu}{t(t+\mu)}\sum_{i=1}^{n} i\left(\frac{t}{t+\mu}\right)^i
\end{aligned}
$$

Hence,

$$t^n(t+\mu) = (t+\mu)^{n+1} - t(\mu(n+1)+t)$$

The greater the value of n, the more deterministic the timeout becomes. As n increases above 1 (the exponential case above), the total timeout rate will increase, tending to a value of around 9 when $\mu = 10$. This corresponds to the upper bound (low arrival rate) of the optimal timeout (optimised for average queue size) found numerically with bounded queues.

Now consider the case where the queues are bounded. Jobs lost on arrival at node 1 do not demand service, however jobs lost at node 2 will cause a relative reduction in demand at node 2 and so must be considered. Thus, in order to balance the service demands it is necessary to decrease the timeout duration (increase t).

For convenience, approximate node 2 as an M/M/1/K_2 queue with average arrival rate λ_2 and average service rate $\mu_2 = (t + sn)/st$. To calculate λ_2 it is necessary to model node 1 to obtain the job loss rate l and the rate of timeout. Approximate node 1 as an M/M/1/K_1 queue with with average arrival rate λ and average service rate μ_1, given by,

$$\frac{1}{\mu_1} = \left(\frac{t}{t+n}\right)^n \frac{n}{t+s} + \frac{s}{t(t+s)} \sum_{i=1}^{n} n \left(\frac{t}{t+s}\right)^i$$

This gives l, hence,

$$\lambda_2 = (\lambda - l)\left(\frac{t}{t+s}\right)^n$$

Thus the loss rate and workload at node 2 are easily estimated and an optimal value of t can be found.

Clearly, the optimal value of t will depend on what metric it is being optimised against. Exactly the same procedure can be employed to estimate the value of t that optimises throughput (minimises job loss). The case where the service demands are hyper-exponential is rather more complex, but a similar argument can be followed. Obviously the residual service of the hyper-exponential will have a longer average than the original service demand, since the proportion of longer jobs will be greater. Thus to balance the service demand across the two nodes far more jobs will need to be processed at node 1 than in the exponential case. This is clearly seen in the following numerical results.

5 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 6 shows the effect of the timeout rate on the average queue size (in total and for each individual queue), plots are also included for random job assignment and the (optimal) shortest queue assignment strategies by way of contrast. The average total timeout duration in each case is simply n/t.

Figure 7 shows the average response times for the same systems.

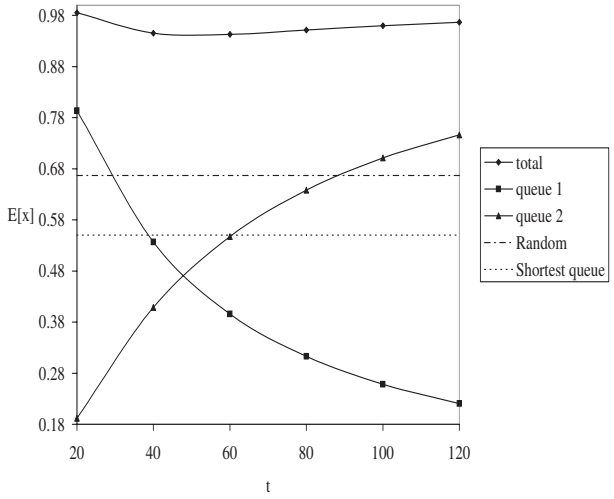

Figure 6. Average queue length varied against timeout rate, $\lambda = 5$, $\mu = 10$

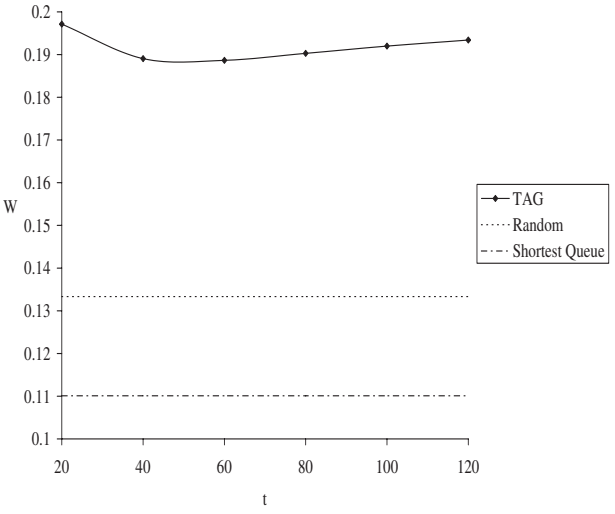

Figure 7. Average response time varied against timeout rate, $\lambda = 5$, $\mu = 10$

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 shape of the curves for TAG in Figures 6 and 7.

In Figure 8 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 λ =11, 9, 7 and 5 respectively), corresponding to average total timeout durations of approximately 7, 7.5, 8.17 and 8.67 respectively. Again these are compared with the average response times for random allocation and the shortest queue

strategy.

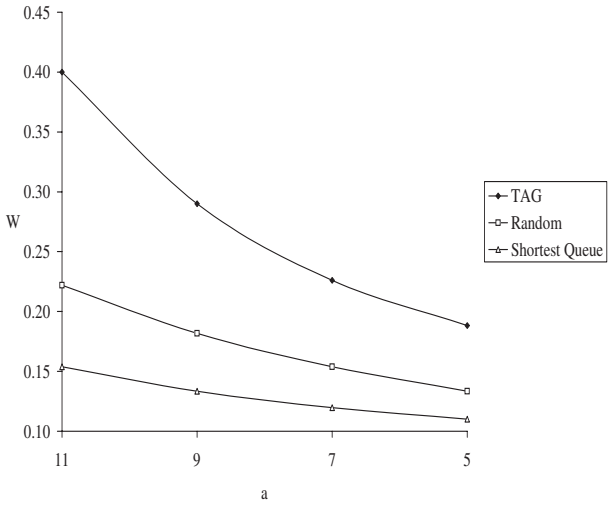

Figure 8. Average response time varied against arrival rate, $\mu = 10$

It would appear from these results that TAG isn't very good compared with 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. Figure 9 shows the average response time varied against timeout rate when the service demand has an H_2 distribution. Results are shown for TAG and shortest queue, where $\alpha = 0.99$, in each case the average service demand is 0.1 and $\mu_1 = 100\mu_2$. All other parameters remain as previously.

In this case random allocation works poorly ($W > 1$, not shown), but TAG is shown to outperform the shortest queue strategy for a wide range of values of t. It should also be remembered that TAG assumes no knowledge of the incoming jobs or the state of the queues, as such it has a lower overhead than the shortest queue strategy. It is also interesting to note that the optimal value of t here is very different from that for the exponential service demand with the same mean, shown in Figure 6. In the exponential case the optimal timeout is much shorter; this is because there are relatively few long jobs, so to balance the workload more short jobs must be pushed through to the second node. In this hyper-exponential case, only 1% of jobs are long, but they are on average 100 times longer than the shorter jobs. Hence it is advantageous to process as many short jobs at the first node as possible, in order to leave the second node free to process the longer jobs.

TAG will give different optimal values of t according to

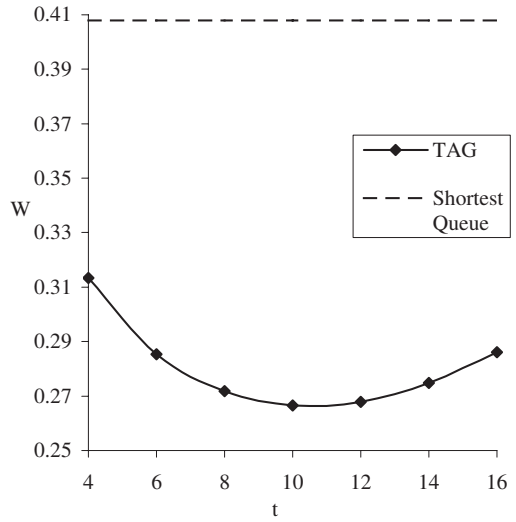

Figure 9. Average response time length varied against timeout rate, $\lambda = 11, \mu = 10, \alpha = 0.99$

what metric is being considered, because of its unusual structure. 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 [5], where different optimisations are presented for slowdown, waiting time and fairness.[1] This is further illustrated in Figure 10 where the same parameters are used as Figure 9, but the metric of interest is throughput.

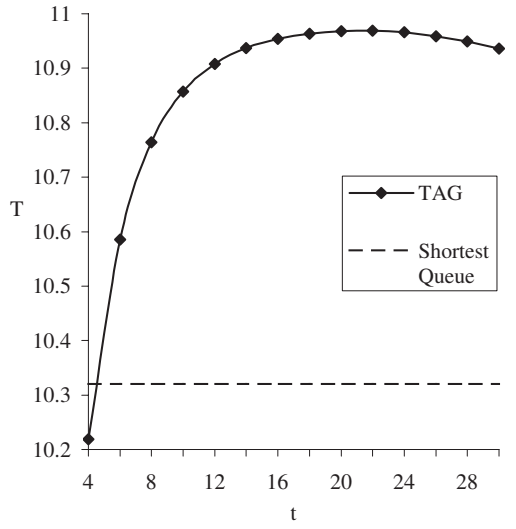

Figure 10. Throughput varied against timeout rate, $\lambda = 11, \mu = 10, \alpha = 0.99$

TAG clearly out performs the shortest queue strategy when reasonably close to optimal t. However, it is also quite sensi-

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

tive to t, and when poorly tuned (e.g. $t = 4$) the throughput falls significantly and the shortest queue strategy will be better. This indicates that there may be scenarios where TAG performs only marginally better than other methods and the therefore the cost of optimising the timeout value (for varying demands, say) may out weigh the limited performance gains.

The reason why a well tuned TAG generally performs so much better than the shortest queue strategy is simple to explain. The shortest queue strategy will lose jobs when both queues are full. This will be most likely to happen when there are two long jobs in the system; one arrives and is sent to one queue, this occupies the server and so if another long job arrives it is more likely to enter the other queue. In such a situation both servers are occupied but jobs continue to arrive, eventually leading to both queues becoming full. In contrast, TAG will lose jobs when either of the queues are full, however, neither situation is particularly likely unless the load becomes excessive. The first queue is unlikely to become full as no job will spend long in service, due to the timeout mechanism. Despite this relatively few jobs will proceed to the second node, so although these jobs remain for a long period, there are generally too few of them to cause the queue to overflow.

The parameters of the hyper-exponential distribution used in Figures 8 and 9 are deliberately extreme, although they broadly correspond to parameters of the bounded Pareto distribution used in [5], which were based on observed traffic. In Figures 11 and 12 long jobs take on average ten times longer than short jobs, i.e. $\mu_1 = 10\mu_2$, rather than one hundred times as previously. The proportion of short jobs is also varied, from $\alpha = 0.89$ to $\alpha = 0.99$. In each case only the optimal value of t is used.

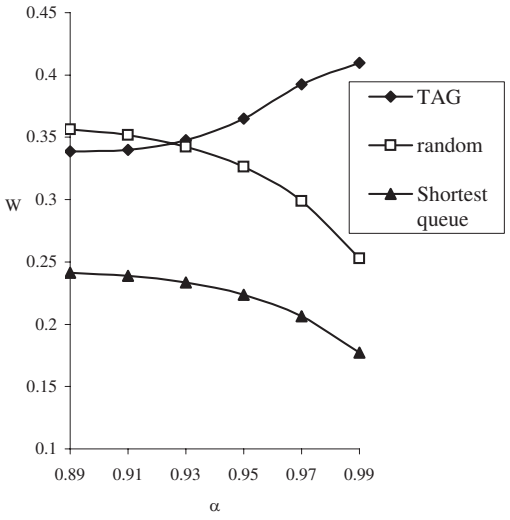

Figure 11. Average response time varied against proportion of longer jobs, $\lambda = 11, \mu = 10$**,** $\mu_1 = 10\mu_2$

Figures 11 and 12 show that the response time increases and the throughput decreases under TAG as α increases.

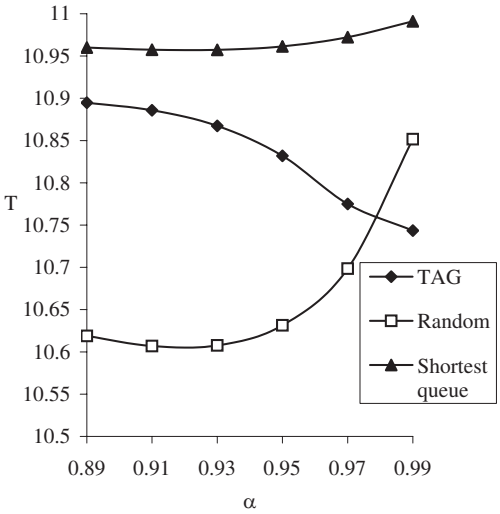

Figure 12. Throughput time varied against proportion of longer jobs, $\lambda = 11, \mu = 10$**,** $\mu_1 = 10\mu_2$

There is a slight levelling off of these curves as α approaches 0.99. Both random allocation and the shortest queue strategy show the reverse trend for each metric. In the case of random allocation the effect of decreasing the proportion of longer jobs to $\alpha = 0.99$ dramatically increases the performance. These results may look confusing, although in fact the explanation is simple. As α increases there are fewer long jobs, and so although their average length increases (in order to satisfy the average service demand being constant and $\mu_1 = 10\mu_2$), there is a reduced probability that both servers will be busy serving long jobs under random allocation or the shortest queue strategy. Clearly this means that these strategies will perform better as $\alpha \longrightarrow 0.99$. Obviously $\alpha = 1$ is simply the exponential case observed in earlier figures, where both these strategies out-performed TAG. As α decreases, the total service demand made by short jobs approaches the total service demand for long jobs. In this situation TAG becomes more efficient as the balance of jobs between the nodes becomes optimal.

6 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 [2], Herzog and Mertsiotakis [6] and Thomas and Hillston [10]. Thomas and Hillston [10] 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.

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 [5]. Related earlier studies were conducted by Crovella et al [4] and Bestavros [3]. In each of these cases it was assumed that a job could not be inter-

rupted 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.

To the knowledge of the author nobody has yet 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 [11].

7 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. The degree of error introduced by these approximations has not been investigated in this paper, but is left for future work. It has been assumed that the job services are exponentially distributed. It is also possible to model phase type distributions, and an approach using hyperexponential distributions is described.

TAG has been shown to offer a significant improvement over conventional mechanisms used for load balancing when the service demand has high variance. This has been demonstrated here for average queue size, response time and throughput, whereas earlier studies concentrated on *mean slowdown* [5]. In such situations TAG out performs the shortest queue policy, which requires knowledge of queue sizes at arrival instants. However, there is an additional overhead in determining an optimal timeout value. This value has been shown to be sensitive to the distribution of the service demand and parameters of the arrival stream. Failure to optimise the timeout may lead to TAG performing worse than other strategies.

There are two main assumptions that enforce the observations made in this paper. Firstly, it has been assumed that the storage demand of a job is approximately constant regardless of its service demand. This means that the capacity to store jobs is considered in the same way at both nodes. Additionally, the arrival process has been assumed to be a Poisson stream. It is expected that TAG would perform less well if the arrival process was bursty. If bursts consisted solely of short jobs then this would affect TAG more than the shortest queue strategy, as shortest queue would share the burst load, whereas TAG would direct all traffic to node 1. In such a scenario TAG might potentially be improved by having a dynamic timeout duration that adapts to queue length or arrival rate. This remains an area of future investigation.

Acknowledgements

The author would like to thank Dr Uli Harder of Imperial College London for introducing him to this area. The author is supported by an EPSRC funded project on *Dynamic Operating Policies for Commercial Hosting Environments* (EP/C009797/1).

References

[1] S. Asmussen, O. Nerman and M. Olsson, Fitting phase-type distributions via the em algorithm, *Scandinavian Journal of Statistics*, 23, pp 419-441, 1996.

[2] M. Bernardo, L. Donatiello and R. Gorrieri, Describing Queueing Systems with MPA, Technical Report UBLCS-94-11, University of Bologna, May, 1994.

[3] A. Bestavros, Load profiling: A methodology for scheduling real time tasks in a distributed system, in: *Proceedings of the 17th International Conference on Distributed Computing Systems, ICDCS '97*, IEEE Computer Society, 1997.

[4] M. Crovella, M. Harchol-Balter and C. Murta, Task assignment in a distributed system: Improving performance by unbalancing load (extended abstract), in: *Proceedings ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Poster Session*, Performance Evaluation Review, **26**(1), 1998.

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

[6] 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*, Erlangen, 1994.

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

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

[9] 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.

[10] 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.

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

Appendix A: Model of a weighted random allocation strategy

$$
\begin{array}{rcl}
Queue1_0 & \stackrel{def}{=} & (arrival1, \lambda_1).Queue1_1 \\
Queue1_j & \stackrel{def}{=} & (arrival1, \lambda_1).Queue1_{j+1} \\
 & & +(service1, \mu_1).Queue1_{j-1} \\
 & & , \ 1 \leq j \leq N-1 \\
Queue1_N & \stackrel{def}{=} & (service1, \mu_1).Queue1_{N-1} \\
 & & \\
Queue2_0 & \stackrel{def}{=} & (arrival2, \lambda_2).Queue2_1 \\
Queue2_j & \stackrel{def}{=} & (arrival2, \lambda_2).Queue2_{j+1} \\
 & & +(service2, \mu_2).Queue2_{j-1} \\
 & & , \ 1 \leq j \leq N-1 \\
Queue2_N & \stackrel{def}{=} & (service2, \mu_2).Queue2_{N-1}
\end{array}
$$

$$Queue1_0 || Queue2_0$$

Figure 13. A PEPA model of two queues in parallel ($\lambda = \lambda_1 + \lambda_2$)

Appendix B: Model of the shortest queue strategy

$$
\begin{array}{rcl}
Queue1_0 & \stackrel{def}{=} & (arr1, \top).Queue1_1 \\
Queue1_j & \stackrel{def}{=} & (arr1, \top).Queue1_{j+1} \\
 & & +(serv1, \top).Queue1_{j-1} \\
 & & , \ 1 \leq j \leq N-1 \\
Queue1_N & \stackrel{def}{=} & (serv1, \top).Queue1_{N-1} \\
 & & \\
Queue2_0 & \stackrel{def}{=} & (arr2, \top).Queue2_1 \\
Queue2_j & \stackrel{def}{=} & (arr2, \top).Queue2_{j+1} \\
 & & +(serv2, \top).Queue2_{j-1} \\
 & & , \ 1 \leq j \leq N-1 \\
Queue2_N & \stackrel{def}{=} & (serv2, \top).Queue2_{N-1} \\
 & & \\
S_0 & \stackrel{def}{=} & (arr1, \lambda_1).S_1 \\
 & & +(arr2, \lambda_2).S_{-1} \\
 & & +(serv1, \mu_1).S_{-1} \\
 & & +(serv2, \mu_2).S_1 \\
S_j & \stackrel{def}{=} & (arr2, \lambda_1 + \lambda_2).S_{j-1} \\
 & & +(serv1, \mu_1).S_{j-1} \\
 & & +(serv2, \mu_2).S_{j+1} \\
 & & , \ 1 \leq j \leq N \\
S_j & \stackrel{def}{=} & (arr1, \lambda_1 + \lambda_2).S_{j+1} \\
 & & +(serv1, \mu_1).S_{j-1} \\
 & & +(serv2, \mu_2).S_{j+1} \\
 & & , \ -N \leq j \leq -1
\end{array}
$$

$$(Queue1_0 \parallel Queue2_0) \underset{(arr1, arr2, serv1, serv2)}{\bowtie} S_0$$

Figure 14. A PEPA model of two $M/M/1/N$ balanced queues in parallel