Comparing job allocation schemes where service demand is unknown

Nigel Thomas

School of Computing Science, University of Newcastle, UK

Received 1 November 2006; received in revised form 1 March 2007
Available online 20 July 2007

Abstract

In this paper a novel job allocation scheme in distributed systems (TAGS) 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 TAGS can perform well for a range of performance metrics. Furthermore, an attempt is made to characterise those scenarios where TAGS is beneficial in terms of the coefficient of variation and load.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Load balancing; Scheduling; Performance modelling; Stochastic process algebra

1. Introduction

Ideally in job allocation 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. This requires the scheduler to have accurate and up to date knowledge of all the queue sizes. As such this strategy cannot be considered as local scheduling, but it is used here for comparison with TAGS.
- Assign jobs to service centres on a rotational basis. This is a very simple strategy and will be fair if the service demands are fairly constant and the servers are homogeneous.
- Assign jobs to service centres randomly according to probabilities based on service capacity. This is another simple strategy, with the advantage that it can balance load in a heterogeneous system. This strategy is also used as a comparison with TAGS.

This final option is illustrated in Fig. 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.

Harchol-Balter [6] introduced a new algorithm (known as Task Assignment based on Guessing Size or TAGS) 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. The system is illustrated in Fig. 2.

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. However, the checkpointing required to allow resumption of service at the next node may in itself contribute a significant delay. Therefore, such a mechanism will adversely affect short jobs in order to reduce the service time of longer jobs.

The efficiency of TAGS 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 ahead of them
in the queue. Obviously 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 tightly 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 [6] 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, $\bar{S}$. Slowdown is defined as follows:

$$S_i = \frac{W_i}{\mu_i}$$

where $W_i$ is the response time for the $i$th job and $s_i$ is its service demand.

The aim of this paper is to model the TAGS algorithm using the Markovian process algebra PEPA, to investigate the
effectiveness of TAGS when the queues are finite and to determine what factors influence the efficiency of TAGS. 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. This 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 magni-
tude. 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. The
grid based SkyServer astronomical database employs a scheme similar to TAGS.

In this paper the queues are bounded, jobs can therefore be lost from the system, either by being dropped at the
first node on arrival, or at subsequent nodes after completing a timed-out service at the predecessor. This latter case
is particularly important as it means that work is being undertaken on jobs which are subsequently 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, as well as average response
time, as an important measure 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 next section gives an overview of some related work. The subsequent section briefly introduces PEPA. This
is followed by a PEPA model of the TAGS algorithm, an alternative model and some notes on using phase type
distributions and some simple approximations. Section 5 presents some numerical results, which is followed by some
conclusions and opportunities for future work.
2. 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 [7] and Thomas and Hillston [12]. Thomas and Hillston [12] covered a number of different queueing scenarios and showed how PEPA could be used to model these. The model specified in Fig. 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 [6]. Related earlier studies were conducted by Crovella et al. [5] and Bestavros [3]. 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.

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 [16].

3. PEPA

A formal presentation of PEPA is given in [8], 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).

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 \((\alpha, r)\) is described by the type of the activity, \(\alpha\), 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:

\[
P ::= (\alpha, r).P \mid P + Q \mid P / L \mid P \triangleleft\triangleleft_Q L A.
\]

- The component \((\alpha, r).P\) performs the activity of type \(\alpha\) 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 \(\tau\).
- Concurrent components can be synchronised, \(P \triangleleft\triangleleft_Q L Q\), 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 \(\parallel\) is used as shorthand to denote synchronisation with no shared activities, i.e. \(P \parallel Q \equiv P \triangleleft\triangleleft_Q L\).
- \(A \overset{\text{def}}{=} P\) 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 \(P \triangleleft\triangleleft_L Q\). 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 (see [8]).

4. The model

The TAGS system is now modelled in PEPA. Since PEPA is Markovian, it is necessary to model the TAGS time-outs 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 (introduced in Section 4.2). Queues are modelled in a state based fashion, depicting each number of jobs in a queue as a separate named derivative of the queue.
enabling the high degree of accuracy in a steady state solution. Prior to service, the timeout clock must be started, this is done by ample). The deterministic timeout at the first node is modelled as an Erlang process with arbitrary number of ticks scheme for a two node system.

At the second node the amount of service from the timeout period must be repeated, this is included in the action repeatservice. Such a mechanism could be modelled by removing the this job until it completes or an arrival occurs, if the queue is otherwise empty, however this is not part of the TAGS algorithm. Such a mechanism could be modelled by removing the timeout action from Queue11 and introducing a separate arrival action in Queue11 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 at rate \( \mu \) 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. The model illustrated in Fig. 3 gives rise to a CTMC with \( K_1(n+1)+1 \) states.

component. An alternative specification of the queues is given in Section 4.1. Figure 3 shows a model of the TAGS scheme for a two node system.

This model uses a few common mechanisms that are familiar from other SPA queueing papers (see [12] for example). The deterministic timeout at the first node is modelled as an Erlang process with arbitrary number of ticks (as is conventional). Earlier studies, e.g. [13], have shown that six phases of an Erlang process are sufficient to give a high degree of accuracy in a steady state solution. 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 3.2). If service1 wins the race then the job departs, otherwise it proceeds to Node2. 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 TAGS algorithm. Such a mechanism could be modelled by removing the timeout action from Queue11 and introducing a separate arrival action in Queue11 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 at rate \( \mu \) 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. The model illustrated in Fig. 3 gives rise to a CTMC with \( K_1(n+1)+1 \) states.
\[ Q_0 \overset{\text{def}}{=} (\text{arrival}, \top).Q_{11}, \]
\[ Q_1 \overset{\text{def}}{=} (\text{timeout}, \top).Q_0 + (\text{service}1, \top).Q_0 + (\text{tick}1, \top).Q_1, \]
\[ Q_{20} \overset{\text{def}}{=} (\text{timeout}, \top).Q_{21}, \]
\[ Q_{21} \overset{\text{def}}{=} (\text{service}2, \top).Q_{20} + (\text{tick}2, \top).Q_{21}. \]
\[ S_1 \overset{\text{def}}{=} (\text{arrival}, \lambda).S_1 + (\text{service}1, \mu).S_1, \]
\[ S_2 \overset{\text{def}}{=} (\text{repeat} \text{service}, \top).S_2 \]
\[ \text{Queue}_1 \overset{\text{def}}{=} S_1 \overset{\text{arrivals}}{\ni} (Q_1 \ldots | Q_1), \]
\[ \text{Queue}_2 \overset{\text{def}}{=} S_2 \overset{\text{service}1}{\ni} (Q_2 \ldots | Q_2), \]
\[ \text{Node}_1 \overset{\text{def}}{=} \text{Timer}_1 \overset{\text{timeout}}{\ni} \overset{\text{service}1, \text{tick1}}{\ni} \text{Queue}_1, \]
\[ \text{Node}_2 \overset{\text{def}}{=} \text{Timer}_2 \overset{\text{repeat} \text{service}}{\ni} \overset{\text{service}2, \text{tick}2}{\ni} \text{Queue}_2. \]

Fig. 4. An alternative PEPA model of a two node system employing TAGS.

4.1. An alternative model

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 Fig. 3 would be modelled as shown in Fig. 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 Fig. 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 [9] and is supported by the Dizzy tool [10]. 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 \( Q_{10} \) 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).

4.2. Phase-type distributions

The exponential distribution is not the most interesting to employ when considering the TAGS 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 TAGS is always going to be suboptimal with exponential arrivals, although it should be remembered that the shortest queue strategy relies on global information, i.e. the router needs to know all the queue lengths. However, TAGS 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 which have been widely used as approximations in many practical situations.

Phase type distributions are distributions constructed by combining multiple exponential random variables. These can be used to approximate most general distributions and extremely accurate approximations can be constructed using tools such as \( EMpht \) [1]. The Erlang distribution, used in Fig. 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, $\alpha$, $\mu_1$ and $\mu_2$ and the following cumulative distribution function,

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

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. The variability of these distributions is generally referred to using the coefficient of variation; defined as the standard deviation divided by the mean. For the exponential distribution this is clearly 1, for an Erlang distribution it is less than 1 and for a hyper-exponential it is greater than 1.\footnote{The Erlang distribution is a special case of the hypo-exponential. It is this relation of the coefficient of variation that gives rise to the hypo-(less) and hyper-(more)exponential distributions being so-called.} The hyper-exponential distribution has successfully been used to model heavy tailed distributions [11], in particular the Pareto distribution used in [6], although it cannot in itself be heavy tailed. As such the hyper-exponential would be ideal for using as a service distribution with the TAGS algorithm. In this paper only the $H_2$ distribution is used, rather than any more complex version of the hyper-exponential, as this is sufficient to demonstrate the features of the model and the TAGS scheme without unduly increasing the state space. A more complex phase type distribution would clearly add greatly to the number of states in the model and consequently add to the difficulty of extracting a numerical solution.

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 $\alpha', \mu_1$ and $\mu_2$. Modelling the $H_2$ distribution in PEPA requires the introduction of some additional terms in order to generate the necessary probabilistic branching. Thus each service action must occur twice, with the rates multiplied by $\alpha$ and $1 - \alpha$, in order to determine whether the next job will be served at rate $\mu_1$ or $\mu_2$. The following modifications (shown in Fig. 5) are made to the basic TAGS model (all other components remain as previously specified).

It is important to note that the Server component will always first perform a service action at rate $\mu_1$ before having the opportunity to branch. Subsequent service actions will occur at either rate according to the branching probability $\alpha$. This behaviour does not affect the overall steady state solution used in this paper, however it should be noted if transient or passage time analysis was applied to this model.

It is worth observing at this point that the claim in [6] is that TAGS works for heavy-tailed service demands. Heavy tailed distributions are often characterised by a so-called power law, namely,

$$\Pr[X > x] \sim c x^{-\alpha}, \quad 1 < \alpha < 2.$$

The decay of a phase type distribution is controlled by one or more exponential distributions, hence,

$$\Pr[X > x] \sim c_1 e^{c_2 x}.$$

As such, a phase type distribution is not heavy tailed, although it may have a large coefficient of variation. It is not possible to model heavy tailed distributions exactly using PEPA.

5. Simple approximations

A key factor in deploying TAGS 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 in a homogeneous system will be found when the average service demand at all nodes is the same. 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,

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

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,

$$\left(\frac{t}{t+\mu}\right)^n \frac{1}{\mu} = \frac{\mu}{(t+\mu)} \frac{1}{(t+\mu)} + \frac{\mu}{(t+\mu)} \frac{t}{(t+\mu)} \frac{2}{(t+\mu)} + \cdots + \frac{\mu}{(t+\mu)} \left(\frac{t}{t+\mu}\right)^{n-1} \frac{n}{(t+\mu)}.$$  

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 $\lambda_2$ and average service rate $\mu_2 = (t+sn)/st$. To calculate $\lambda_2$ 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 average arrival rate $\lambda$ and average service rate $\mu_1$, given by,

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

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.

6. Numerical results

The models presented here have been analysed numerically using the PEPA Workbench Java Edition Tabasco/Release version 0.9.4b, employing the linear biconjugate gradient method. The maximum number of iterations was set to 1000 and the tolerance was set to $10^{-25}$. In all cases the maximum queue length was set to 10 and the number of phases of the Erlang timeout was 6. Even with these relatively small values the PEPA Workbench has some problems deriving the state space and finding the steady state solution to models involving hyper-exponential service, despite the state space being only of the order of 10,000 states.

6.1. Performance of TAGS under exponential service demands

The model specified in Fig. 3 is analysed with $n=5$ and $K_1 = K_2 = 10$. This gives rise to a model of 4331 states. Figure 6(a) 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 6(b) shows the average response time for the same system.

It is important to note that whereas the shortest queue strategy has almost negligible loss at this arrival rate, the random assignment and TAGS 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 TAGS in Fig. 6.

In Fig. 7 the same system is shown with varying arrival rates. The TAGS algorithm is optimised for minimum queue length, the optimal (integer) values of $t$ being 42, 45, 49 and 51 (for $\lambda = 11, 9, 7$ 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.

6.2. Hyper-exponential service demands

It would appear from these results that TAGS is not 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 TAOS has a greater effect. This should not be a great surprise as it is well known that the shortest queue strategy is the optimal

![Fig. 6. (a) average queue length and (b) average response time, varied against timeout rate, $\lambda = 5$, $\mu = 10$.](image)
Fig. 7. Average response time varied against arrival rate, $\mu = 10$.

Fig. 8. (a) average response time and (b) throughput, varied against timeout rate, $\lambda = 11, \mu = 10, \alpha = 0.99$.

task assignment strategy for exponential arrivals and service demands. However, for a service demand with greater variance, TAGS would be expected to perform better. Figure 8(a) shows the average response time varied against timeout rate when the service demand has an $H_2$ distribution. Results are shown for TAGS 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 TAGS is shown to outperform the shortest queue strategy for a wide range of values of $t$. It should also be remembered that TAGS 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 Fig. 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.
TAGS will give different optimal values of $t$ according to 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 [6], where different optimisations are presented for slowdown, waiting time and fairness. This is further illustrated in Fig. 8(b) where the same parameters are used as Fig. 8(a), but the metric of interest is throughput.

TAGS clearly out performs the shortest queue strategy when reasonably close to optimal $t$. However, it is also quite sensitive 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 TAGS 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 TAGS 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, TAGS 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 Fig. 8 are deliberately extreme, although they broadly correspond to parameters of the bounded Pareto distribution used in [6], which were based on observed traffic. In Fig. 9 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 9 shows that the response time increases and the throughput decreases under TAGS as $\alpha$ increases. There is a slight levelling off of these curves as $\alpha$ 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 $\alpha$ increases there are fewer long jobs and so, although the average length of long jobs

---

2 The optimisation for fairness seeks to make the slowdown nearly constant regardless of job 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 $\alpha \to 0.99$. Obviously $\alpha = 1$ is simply the exponential case observed in earlier figures, where both these strategies out-performed TAGS. As $\alpha$ decreases, the total service demand made by short jobs approaches the total service demand for long jobs. In this situation TAGS becomes more efficient as the balance of jobs between the nodes becomes optimal.

6.3. Characterising where TAGS is beneficial

It has been demonstrated above that a hyper-exponential service demand can both benefit, and be penalised, by the use of TAGS, relative to other common strategies, depending on the parameters used. An important question therefore is how can the scenarios be characterised where TAGS is beneficial? One significant factor is the coefficient of variation of the service distribution. The coefficient of variation is defined as follows:

$$cv = \frac{\sigma}{\mu}$$

where $\sigma$ is the standard deviation and $\mu$ is the mean.

In Figs. 6 and 7 the service demands are exponentially distributed, therefore the coefficient of variation is 1. Here TAGS performs poorly compared with the shortest queue and random allocation strategies. In Fig. 8, where $\mu_1 = 100\mu_2$ and $\alpha = 0.99$, the coefficient of variation is approximately 7.07. Here TAGS performs much better than the shortest queue and random allocation strategies (as long as the timeout is optimised). Finally, in Fig. 9, where $\mu_1 = 10\mu_2$ and $\alpha$ is varied between 0.89 and 0.99, the coefficient of variation varies between a minimum of 1.533 ($\alpha = 0.99$) and a maximum of 2.247 ($\alpha = 0.91$). Here TAGS performs poorly compared with the shortest queue strategy, but better than the random strategy when $\alpha < 0.925$ (for average response time); this corresponds to $cv > 2.237$.

These results imply that at some value of $cv$ between those used in Fig. 8 ($cv = 7.07$) and Fig. 9 ($cv = 2.47$), TAGS and the shortest queue strategy should perform very similarly. Figure 10 shows the average response time for a two node system ($n = 5$ and $K_1 = K_2 = 5$) where $\mu_1 = 50\mu_2$ and $\alpha$ is varied between 0.94 ($cv = 4.295$) and 0.7 ($cv = 2.256$).

Figure 10 shows that, with these values of $cv$, TAGS outperforms the random allocation strategy, as we would expect. It also shows that when $\alpha \geq 0.78$ TAGS outperforms the shortest queue strategy, but when $\alpha \leq 0.74$ the shortest queue strategy is best. The point where the two lines cross is $\alpha = 0.7634$, or $cv = 2.543$. Similar results can be derived for throughput.
7. Conclusions and further work

A model of a novel allocation strategy has been presented in this paper using the stochastic process algebra 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. However, earlier use of Erlang distributions in PEPA [4,13] has shown that minimal error is introduced to steady state metrics when $k = 6$. If transient analysis were to be performed on these models then a larger value of $k$ might be required.

It has been assumed that the job services are exponentially distributed. It is also possible to model phase type distributions, and an approach using hyper-exponential distributions is described. No attempt has been made to validate these assumptions against implementation or “realistic” simulation, instead this task has been left for future study. In particular it would be interesting to explore further the issue of heavy tailed distributions and suitable approximations.

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 TAGS would perform less well if the arrival process was bursty. If bursts consisted solely of short jobs then this would affect TAGS more than the shortest queue strategy, as shortest queue would share the burst load, whereas TAGS would direct all traffic to node 1. In such a scenario TAGS 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.

TAGS has been shown to offer a significant improvement over conventional mechanisms used for load balancing when the service demand has high variance (as opposed to the heavy tail assumption made in [6]). This has been investigated here for average queue size, response time and throughput, whereas earlier studies concentrated on mean slowdown [6]. In such situations TAGS 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 TAGS performing worse than other strategies.

The observant reader will have deduced that there are a number of small enhancements that can be made to the TAGS scheme that exploit certain scenarios, for example when one or other of the servers is idle. The most obvious of these is only employing TAGS once a long job has been discovered (it has timed out); the rest of the time the system could run a standard load balancing scheme. However, this enhancement, like others which are apparent, exploits a situation which are rare when the workload is suitable for the deployment of TAGS and so have little effect on the steady state measures considered in this paper. It is however quite possible that such enhancements could be beneficial when considering transient performance measures.

Acknowledgments

The author would like to thank Dr Uli Harder of Imperial College London for introducing him to this area. The author also acknowledges the assistance of Dr Harder and Dr Richard Clegg, of the University of York, for answering some queries regarding heavy tailed distributions. The author is supported by an EPSRC funded project on Dynamic Operating Policies for Commercial Hosting Environments (EP/C009797/1). An earlier versions of this work appeared in [14] and [15].
Appendix A. Model of a weighted random allocation strategy

\[
\begin{align*}
\text{Queue}_1^n & \overset{\text{def}}{=} (\text{arrival}_1, \lambda_1).\text{Queue}_1, \\
\text{Queue}_j^n & \overset{\text{def}}{=} (\text{arrival}_1, \lambda_1).\text{Queue}_{j+1} + (\text{service}_1, \mu_1).\text{Queue}_{j-1}, \quad 1 \leq j \leq N - 1, \\
\text{Queue}_N^n & \overset{\text{def}}{=} (\text{service}_1, \mu_1).\text{Queue}_{N-1}.
\end{align*}
\]

\[
\begin{align*}
\text{Queue}_0^n & \overset{\text{def}}{=} (\text{arrival}_2, \lambda_2).\text{Queue}_2, \\
\text{Queue}_2^n & \overset{\text{def}}{=} (\text{arrival}_2, \lambda_2).\text{Queue}_{2+1} + (\text{service}_2, \mu_2).\text{Queue}_{2-1}, \quad 1 \leq j \leq N - 1, \\
\text{Queue}_N^n & \overset{\text{def}}{=} (\text{service}_2, \mu_2).\text{Queue}_{N-1}.
\end{align*}
\]

\[\text{Queue}_1^n \parallel \text{Queue}_2^n.\]

Fig. 11. A PEPA model of two queues in parallel (\(\lambda = \lambda_1 + \lambda_2\)).

Appendix B. Model of the shortest queue strategy

\[
\begin{align*}
\text{Queue}_1^n & \overset{\text{def}}{=} (\text{arrival}_1, T).\text{Queue}_1, \\
\text{Queue}_j^n & \overset{\text{def}}{=} (\text{arrival}_1, T).\text{Queue}_{j+1} + (\text{service}_1, T).\text{Queue}_{j-1}, \quad 1 \leq j \leq N - 1, \\
\text{Queue}_N^n & \overset{\text{def}}{=} (\text{service}_1, T).\text{Queue}_{N-1}.
\end{align*}
\]

\[
\begin{align*}
\text{Queue}_0^n & \overset{\text{def}}{=} (\text{arrival}_2, T).\text{Queue}_2, \\
\text{Queue}_2^n & \overset{\text{def}}{=} (\text{arrival}_2, T).\text{Queue}_{2+1} + (\text{service}_2, T).\text{Queue}_{2-1}, \quad 1 \leq j \leq N - 1, \\
\text{Queue}_N^n & \overset{\text{def}}{=} (\text{service}_2, T).\text{Queue}_{N-1}.
\end{align*}
\]

\[
\begin{align*}
S_0 & \overset{\text{def}}{=} (\text{arrival}_1, \lambda_1).S_1 + (\text{arrival}_2, \lambda_2).S_{-1} + (\text{service}_1, \mu_1).S_{-1} + (\text{service}_2, \mu_2).S_1, \\
S_j & \overset{\text{def}}{=} (\text{arrival}_1, \lambda_1 + \lambda_2).S_{j-1} + (\text{service}_1, \mu_1).S_{j-1} + (\text{service}_2, \mu_2).S_{j+1}, \quad 1 \leq j \leq N, \\
S_j & \overset{\text{def}}{=} (\text{arrival}_1, \lambda_1 + \lambda_2).S_{j+1} + (\text{service}_1, \mu_1).S_{j+1} + (\text{service}_2, \mu_2).S_{j-1}, \quad -N \leq j \leq -1
\end{align*}
\]

\[
\text{(Queue}_1^n \parallel \text{Queue}_2^n)^{\leq 0} \overset{\text{def}}{=} S_0.
\]

Fig. 12. A PEPA model of two M/M/1/N balanced queues in parallel.

References