Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 353 (2020) 21–38

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Energy Consumption by Servers under Unknown Service Demand

Ali Alssaiari[1] Nigel Thomas[2]

School of Computing
Newcastle University
Newcastle upon tyne, UK

Abstract

We evaluate energy consumption under unknown service demands using three policies: task assignment based on guessing size (TAGS), the shortest queue strategy and random allocation in a homogeneous environment. We modelled these policies using performance evaluation processing algebra (PEPA) to derive numerical solutions. Our results show that servers running under TAGS consumes more energy than other policies in terms of total energy consumption. In contrast, TAGS consumes less energy than random allocation in terms of energy per job when the arrival rate is high and the job size is variable.

Keywords: Energy Consumption, Performance, Modelling, PEPA.

1 Introduction

Interest in computation energy consumption has grown in recent years as we aim to reduce the energy cost of running equipment, such as servers, routers, uninterruptible power sources, cooling systems and other ancillary equipment. However, the use of energy in computer systems continues to grow. Studies show an increased usage of digital electronic devices, such as mobile phones and tablets, with a growing demand for digital media services, such as video streaming and online television services [17]. Furthermore, the role of data centres is expected to develop further in coming years. The internet of things and digital devices rely on data centres as the backbone infrastructure to provide communication services and data transfer services between providers and end-users. It has been reported that data centres alone used 1.5% of global electricity in 2010 [11]. Energy consumption of data centres is expected to increase by 13% from 2010 to 2030 [3]. Energy consumption

[1] Email: a.alssaiari@ncl.ac.uk

[2] Email: nigel.thomas@newcastle.ac.uk

https://doi.org/10.1016/j.entcs.2020.09.017

amounts and techniques for handling electricity usage in data centres have been well-documented in many previous studies, including [16,4,1,8,15,23].

Servers are typically the largest consumers of energy in a data centre [12,18]. However, servers are often used quite inefficiently, consuming energy while the utilisation is low in most of the time [7]. So, studying and investigation job scheduling and allocation policies and techniques in order to overcome the low utilisation is essential. However, consideration should be given to the impact of these policies on energy consumption. Some studies focused on improving system performance without considering the impact of increased performance on energy consumption. Gardner *et al.* [10] introduced a model for job redundancy to decouple server slowdown and job size. The idea of redundancy was to replicate the same job request and send it to more than one server to be processed. Then, when the job was completed on any server, the request was considered to have processed successfully. The primary focus was on reducing response time when the variability in server side was significant. The variability in server refers to the possibility that a server is slow due to high load, network errors or a hardware specification that differs from server to server.

Benjamin *et al.*. [6] focused on studying parallel job scheduling across multiple CPU cores. The main aim of the study was to find a scheduling mechanism to distribute incoming jobs over multi-core in order to reduce the mean response time. Workload modelling and service distribution comparison for modelling a workload have been covered in detail and presented by [9]. Jayasinghe [5] studied and investigated assigning tasks in server farms under realistic workloads. He proposed three different task assignment policies for time sharing in server farms. Moreover, he provided a consideration for scheduling tasks efficiently under heavy-tailed jobs.

In this paper, we consider the impact of the choice of scheduler on performance and energy. We evaluate the task assignment based on guessing size (TAGS) assignment policy and compare it to the shortest queue strategy and random allocation in terms of total energy consumption and energy consumption per job. All the three policies assume no knowledge about service demand or job size. Previously, these policies were evaluated in terms of performance in [20], but no consideration for energy consumption has been given.

The rest of this paper is organised as follows. In Section 2, we analysed and modelled TAGS using the Markovian process algebra PEPA. In Section 3, the numerical results to evaluate the performance metrics are presented. Section 4 introduces the energy model and the energy consumption results. Finally, Section 5 contains a discussion of future work and concludes the paper.

2 The Model

This section illustrates the random allocation policy, the shortest queue strategy and TAGS algorithm. We modelled the three policies in PEPA, a formal presentation of which can be found in [14]. We assume that in all cases, jobs arrive into the system in a Poisson stream and receive a single service before leaving the system.

Jobs are assumed to be independent and identically distributed. We studied energy consumption and performance under two service demand types: (i) exponential service demand and (ii) a two phase hyper-exponential service demand. The queues are bounded as PEPA (with a maximum capacity of 10 jobs) does not support infinite queues. As a result, queues can be full leading to rejecting new arrival jobs from joining the queue. We assume that all queueing is First-Come-First-Served. In Section 2.3 we present the PEPA specification of the model and its parameters.

2.1 Random Allocation

The random allocation policy assigns arrival jobs to a queue randomly. So, it does not take into consideration how many jobs are already waiting in the queue. As a result, one queue might be overflowing with jobs while the other queue is empty or half full. Moreover, the probability of losing jobs is high, as sending a job to a full queue results in dropping that job permanently. Furthermore, short jobs might be delayed for a long time when getting stuck behind a long job, which is not detected by the random scheduler. However, random allocation can be an attractive option as no knowledge about the system is needed. Hence it is relatively trivial to implement.

2.2 The shortest queue

The shortest queue strategy overcomes the problem of load balancing between the queues. When jobs arrive the policy forwards them to the queue with the least waiting jobs, thus leading to no queue becoming full while other queues still have available capacity. So, the probability of losing jobs is less significant as long as the arrival rate does not exceed the system capacity. However, short jobs might be delayed for a long time when getting stuck behind a long job, as this strategy only counts the waiting jobs and not their service demands. The shortest queue involves a management overhead as the policy has to have knowledge of the states of the queues. If the latency in polling queues is long then this overhead might be significant, leading to poorer performance in practice. However, in this paper we do not consider this aspect.

2.3 Task Assignment based on Guessing Size (TAGS)

The TAGS scheme was initially introduced by Harchol-Balter [13] in order to address the problem of jobs with long service demands unduly delaying jobs with a short service. The main justification of this algorithm is to allocate jobs where the service demand is unknown before execution. In this approach, a job is sent to a single server queue. The server starts processing the first job in the queue until the job is completed and has departed successfully or until a fixed time out is reached. If the timeout is reached before the job is completed, then the job is transferred to the next server. When the job arrives at the next server, the same steps are repeated, but the timeout for this level is increased. The process is repeated with a longer timeout each time until the last server is approached. The job in this final stage receives service until completion. It is assumed that there is no checkpointing and so any service accrued at the previous stage must be repeated. Clearly, when compared to the random allocation and the shortest queue, TAGS can overcome the problem of short jobs getting stuck behind a long job, however there is an overhead in the repeated service which can affect performance. If the timeout is too short then too many jobs will require repeated service and if the timeout is too long then the duration of each repeated service will delay completion.

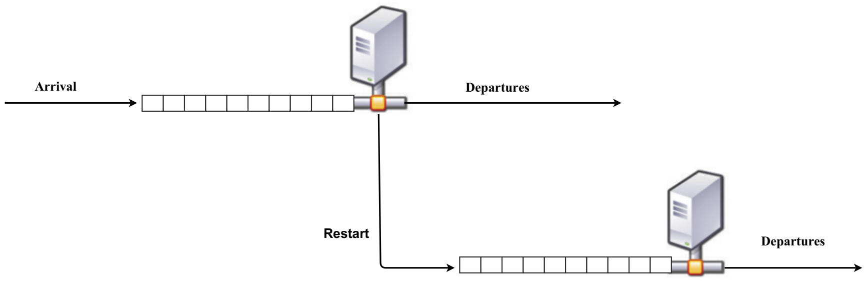


Fig. 1. Jobs allocation flow in TAGS

The main difference between TAGS and multi-level feedback queuing is that in TAGS approach, the job is killed if it reaches the end of the time out period on a server. Afterwards, it is transferred to the next server and starts from the beginning. In contrast, in multi-level feedback queuing, the service resumes on the next server from where it stopped on the previous server. Thus, the effort is not lost, but resource is consumed in recording the execution state, which can be significant. Figure 1 illustrates the concept of TAGS.

Thomas [20] studied and modelled TAGS in PEPA and showed that TAGS could perform well for various performance metrics compared to random allocation and the shortest queue strategy when the job size variability is high; however, the timeout values need to be optimised. In this paper, we consider the energy consumption level and performance using the TAGS scheme and compare it to energy consumption by random allocation and the shortest queue strategy.

2.4 The model in PEPA

The shortest queue and random allocation models in PEPA are depicted in [21] and we used the same TAGS model introduced in [20]. As PEPA is a Markovian Process Algebra, the deterministic timeout used in TAGS is modelled by an Erlang distribution. In each policy the service distribution is considered as either a negative exponential (hence relatively low variance) or a two phase hyper-exponential (relatively high variance). In all cases the inter-arrival periods are negative-exponentially distributed and the maximum queue lengths are finite (maximum size 10 jobs). Figure 2 illustrates the TAGS model in PEPA under exponential demand while Figure 3 illustrate the model under hyper-exponential demand. The notation of the PEPA model are summarised in Table 1.

For numerical tractability and ease of understanding, the number of nodes is restricted to two, as this is sufficient to investigate the consequences of using the

$$
\begin{aligned}
Q1_0 &\overset{def}{=} (arrival, \lambda).Q1_i;\\
Q1_i &\overset{def}{=} (arrival, \lambda).Q1_{i+1} + (service1, \top).Q1_{i-1}\\
&\quad +(tick1, \top).Q1_i + (timeout, \top).Q1_{i-1}; \quad 1 \le i < K1\\
Q1_n &\overset{def}{=} (service1, \top).Q1_{n-1} + (tick1, \top).Q1_n + (timeout, \top).Q1_{n-1}\\
Server1 &\overset{def}{=} (service1, \mu).Server1;\\
Timer1_0 &\overset{def}{=} (timeout, t).Timer1_n + (service1, \top).Timer1_n\\
Timer1_i &\overset{def}{=} (tick1, t).Timer1_{i-1} + (service1 \top).Timer1_n \quad 1 \le i \le n\\
Q2_0 &\overset{def}{=} (timeout, \top).Q2_i\\
Q2_i &\overset{def}{=} (timeout, \top).Q2_{i+1} + (tick2, \top).Q2_i\\
&\quad +(repeatservice, \top).Q2'_i \qquad 1 \le i < K2\\
Q2_{K2} &\overset{def}{=} (timeout, \top).Q2_{K2} + (tick2, \top).Q2_{K2}\\
&\quad +(repeatservice, \top).Q2'_{K2}\\
Q2'i &\overset{def}{=} (timeout, \top).Q2'_{i+1} + (service2, \top).Q2_{i-1} \quad 1 \le i < K2\\
Q2'_{K2} &\overset{def}{=} (timeout, \top).Q2'_{K2} + (service2, \top).Q2'_{K2-1}\\
Timer2_0 &\overset{def}{=} (repeatservice, t).(service2, \mu).Timer2_n\\
Timer2_i &\overset{def}{=} (tick2, t).(Timer2_i) \quad 1 \le i \le n
\end{aligned}
$$

$$((Q1_0 \underset{service1}{\bowtie} Server1) \underset{K}{\bowtie} Timer1_n) \underset{timeout}{\bowtie} (Queue2_0 \underset{L}{\bowtie} Timer2_n)$$

$$where\ K = (service1, timeout, tick1) \text{ and } L = (repeatservice, service2, tick2)$$

Fig. 2. A PEPA TAGS exponential model

Notation	Meaning
$Q1_i$	the first queue.
$Server1$	the first server.
$Timer1_i$	the first timer which governs the decision to terminate the job execution at the first server after a specific processing time.
$Timer2_i$	the timer in the second server used to model the repeated service.
$Q2_i$	the second queue.
$K1$	the maximum length of the first queue.
$K2$	the maximum length of the second queue.
$arrival$	the arrival process.
$service_i$	the service process, $i = 1, 2$.
$timeout$	the timeout action which kills the job at $server1$ when triggered.
$tick1$	the tick action of the timeout clock.
$repeateservice$	the repeat service action, that repeats the amount of service that timed out previously in $server1$.

Table 1
PEPA Model Notation

TAGS scheme on energy consumption. The queue size is bounded, and hence a job can be lost at arrival by being dropped from the first node or at the subsequent node after completing a timeout service. Thus, a proportion of jobs might be lost from the second node if the load is high and, the timeout at the first node is too short. In contrast, a long timeout at the first node will increase the probability of losing a job at the first node as the queue becomes full, rejecting new arrivals from joining the queue.

The queues are modelled in such a way that each job is represented as a separately named derivative of the queue. The timeout at the first node is modelled using an Erlang distribution, and the number of ticks is fixed. While the queue is not empty, the timeout clock starts at the beginning of each derivative of the queue. This is done by introducing *tick* action at each derivative. A race exists between the timeout action and the service process *service*1. If the timeout action wins, the job is killed and transferred to the second node to restart service from the beginning. Otherwise, the task departs the system since it finished before the timeout action triggered. In both situations, the timeout clock is reset. If a job is waiting in the queue, the race starts again; otherwise, if the queue is empty, the server enters an idle state until a new job arrives.

After timing out, the job restarts at the second node and receives a repeat process of the amount of service (the same number of ticks) that timed out in the first node. To overcome the resampling problem, this is represented by introducing the *repeatservice* action in *timer*2, while the remaining part of the job receives service from action *service*2.

2.5 Hyper-exponential distribution

When considering TAGS, the exponential distribution is not the most interesting to use as the main motivation for TAGS is to enable throughput of short jobs in the presence of long running jobs. Hence TAGS will perform best with a mixed workload where there are lots of short jobs and a few very long running jobs.

Modelling the hyper-exponential distribution in PEPA involves the implementation of some extra factors to produce the required probabilistic branching. Each *timeout* and *service*1 action must, therefore, take place twice, with rates multiplied by α and (1- α) to determine whether the next job will be served at the appropriate rate, $\mu1$ or $\mu2$ (in derivatives $Q1_i$ and $Q1'_i$ respectively). The exception is $Q1_1$, if such actions lead to emptying of the queue, where the required branching will be carried out by an arrival action in $Q1_0$. In the second node, the branching process is less complex with the branching taking place at the *repeatservice* action. Clearly the probability that a short job times out will be less than the probability that a long job times out, which necessitates computing the resultant probability α'. Figure 3 shows the TAGS model after the H2 distribution support change has been implemented.

3 Numerical solution

The model was analysed numerically using the PEPA eclipse plug-in [22]. For validity, we used the same model and variables that have been used by [20] for performance analysis. The maximum queue length in all cases was set to be ($K1 = K2 = 10$).The number of phases of the Erlang timeout was 6, n=5. The proportion of short job was set to α= 0.99 and the average service to μ=10. The long job size was set to be 100 times longer than the short job by specifying the service rate at

$$\begin{aligned}
Q1_0 &\overset{def}{=} (arrival, \lambda).Q1_i;\\
Q1_i &\overset{def}{=} (arrival, \lambda).Q1_{i+1} + (service1, \top).Q1_{i-1}\\
&\quad +(tick1, \top).Q1_i + (timeout, \top).Q1_{i-1}; \qquad 1 \le i < K1\\
Q1_n &\overset{def}{=} (service1, \top).Q1_{n-1} + (tick1, \top).Q1_n + (timeout, \top).Q1_{n-1}\\
\\
Server1 &\overset{def}{=} (service1, \alpha * \mu_1).Server1 + (service1, (1-\alpha) * \mu_1).Server1'\\
&\quad +(timeout, \alpha * t).Server1 + (timeout, (1-\alpha) * t).Server1'\\
&\quad +(tick1, t).Server1\\
\\
Server1' &\overset{def}{=} (service1, \alpha * \mu_2).Server1 + (service1, (1-\alpha) * \mu_2).Server1'\\
&\quad +(timeout, \alpha * t).Server1 + (timeout, (1-\alpha) * t).Server1'\\
&\quad +(tick1, t).Server1'\\
Timer1_0 &\overset{def}{=} (timeout, t).Timer1_n + (service1, \top).Timer1_n\\
Timer1_i &\overset{def}{=} (tick1, t).Timer1_{i-1} + (service1, \top).Timer1_n \quad 1 \le i \le n\\
\\
Q2_0 &\overset{def}{=} (timeout, \top).Q2_i\\
Q2_i &\overset{def}{=} (timeout, \top).Q2_{i+1} + (tick2, \top).Q2_i\\
&\quad +(repeatservice, \top).Q2'_i, \qquad 1 \le i < K2\\
Q2_{K2} &\overset{def}{=} (timeout, \top).Q2_{K2} + (tick2, \top).Q2_{K2} + (repeatservice, \top).Q2'_{K2},\\
\\
Q2'i &\overset{def}{=} (timeout, \top).Q2'_{i+1} + (service2, \top).Q2_{i-1}, \quad 1 \le i < K2\\
Q2'_{K2} &\overset{def}{=} (timeout, \top).Q2'_{K2} + (service2, \top).Q2'_{K2-1}\\
\\
Timer2_0 &\overset{def}{=} (repeatservice, \alpha' * t).(service2, \mu_1).Timer2_5\\
&\quad +(repeatservice, (1-\alpha') * t).(service2, \mu_2).Timer2_5\\
Timer2_5 &\overset{def}{=} (tick2, t).Timer2_4\\
Timer2_4 &\overset{def}{=} (tick2, t).Timer2_3\\
Timer2_3 &\overset{def}{=} (tick2, t).Timer2_2\\
Timer2_2 &\overset{def}{=} (tick2, t).Timer2_1\\
Timer2_1 &\overset{def}{=} (tick2, t).Timer2_0
\end{aligned}$$

$$((Q1_0 \underset{K}{\bowtie} Server1) \underset{K}{\bowtie} Timer1_5) \underset{timeout}{\bowtie} (Queue2_0 \underset{L}{\bowtie} Timer2_5)$$
$$where\ K = (service1, timeout, tick1) \text{ and } L = (repeatservice, service2, tick2)$$

Fig. 3. A PEPA TAGS hyper exponential model

the second node to be 100 times less than the service rate at the first node: $\mu_1 = 100\mu_2$. The jobs are independent (i.e., they do not share resources) and have no priority constraints. This gives rise to a model of 4331 states in the exponential case and raises to 9882 states in the hyper exponential case. We did not evaluate a more extensive system, as the CTMC has almost 10000 states, which is close to the maximum number of states we were able to solve using the PEPA Eclipse plug-in.

For all scenarios in this paper TAGS was optimised for the maximum throughput by computing the timeout rate t to the optimal value for each arrival rate.

3.1 Performance analysis under exponential service demand

The model specified in Figure 2 represents the case of exponential demand analysed to obtain the throughput and the average response time. The average timeout duration in each case can be calculated as $t/n + 1$.

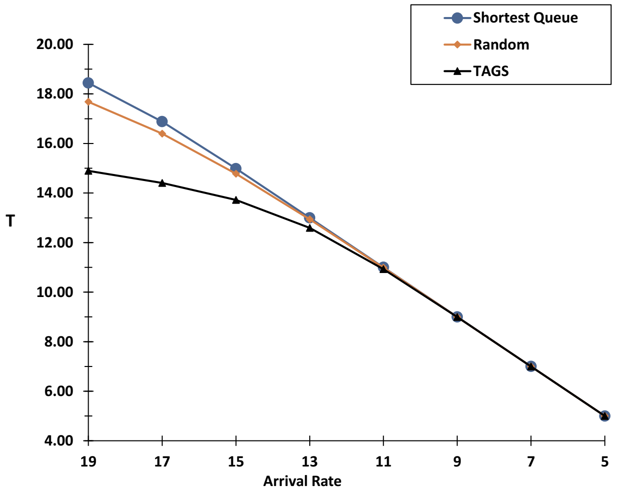


Fig. 4. Throughput varied against arrival rate λ, μ= 10.

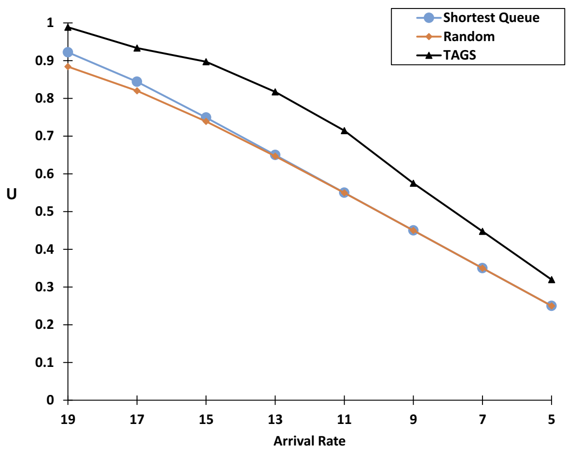
Figure 4 shows the throughput varied against arrival rate λ. The TAGS algorithm is optimised for the maximum throughput, the optimal values of t being 52, 54, 54 and 54 (for λ = 11, 9, 7 and 5, respectively). corresponding to average timeout durations of approximately 8.66, 9, 9 and 9 , respectively. Random allocation and shortest queue plots are included for contrast as well. Figure 5 shows the utilisation of both servers.

The job loss at the shortest queue strategy is nearly insignificant at all arrival rates (0.5 at the highest arrival rate and 10^{-12} at the lowest arrival) , while at the random assignment and the TAGS is slightly higher at high arrival rates.

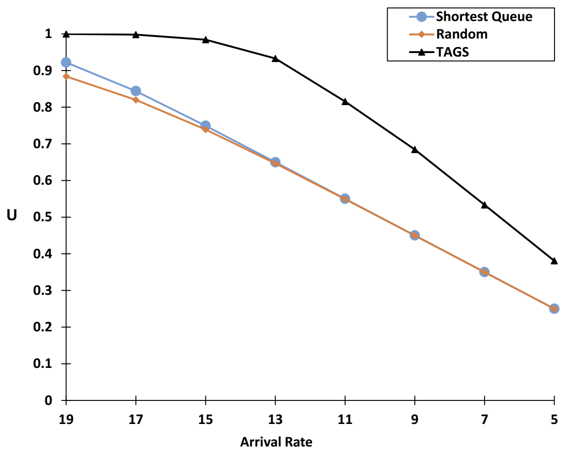
These findings indicate that TAGS is not very useful in comparison with the random and shortest queue strategies under exponential service demand, notably as the service demand increases resulting in an increase in the incomplete jobs rate in TAGS. This is expected as it is well known that the optimal strategy for exponential arrivals and service demands is the shortest queue.

3.2 Performance analysis under hyper-exponential service demand

Hyper-exponential demand has a greater variance in service demand than exponential demand which makes it an appropriate distribution to investigate the performance metrics for TAGS. Figure 6 shows the throughput varied against arrival rate when service demand has an H_2 distribution. Results are shown for TAGS, shortest queue and random allocation. The proportion of short job was set to α= 0.99 and the average service to μ=10. The long job size was set to be 100 times longer than the short job by specifying the service rate at the second node to be 100 times less



(a) Server1



(b) Server2

Fig. 5. (a) $server_1$ utilisation and (b) $server_2$ utilisation, varied against arrival rate λ, $\mu = 10$.

than the service rate at the first node: $\mu_1 = 100\mu_2$.

TAGS obviously outperforms the shortest queue and random allocation when service demand is increased, and load variability is high. The explanation why TAGS is better than shortest queue is easy to clarify. The shortest queue strategy will lose jobs when both queues are occupied by two long jobs Figure 7. This can happen when a one long job arrives to the system and forwarded to the first server and if another long job arrives it will be forwarded to the second server as the first one is already occupied by a long job.

As a result both queues will become full and if any new job arrives to the system will be dropped from the queue leading to an increase in the job loss rate. In contrast, TAGS reduces the chance that both queues become full if the timeout is well-tuned. The first queue is unlikely to become full as the timeout mechanism will kill long jobs and transfer them to the second server. Despite the fact that the processing time for each long job is 100 times longer than any short job, the

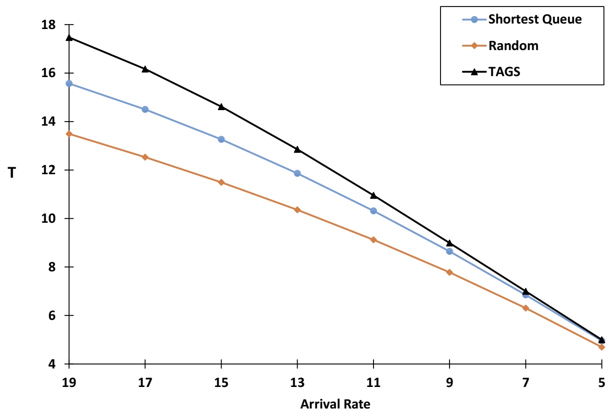


Fig. 6. Throughput varied against arrival rate λ, $\mu = 10$, $\alpha = 0.99$, $\mu_1 = 100\mu_2$

probability of the second queue to become full is relatively small as there are too few long jobs.

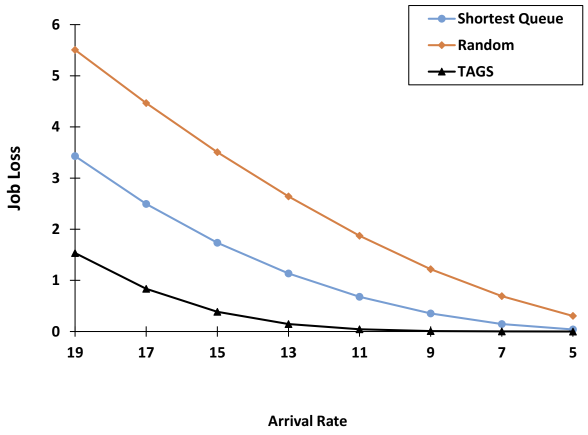


Fig. 7. Job loss varied against arrival rate λ, $\mu = 10$, $\alpha = 0.99$, $\mu_1 = 100\mu_2$

The results become more interesting when we looked at the average response time. Figure 8 shows the average response time varied against arrival rate. It can be seen at low arrival rates ($\lambda < \mu = 10$) TAGS outperforms shortest queue and random allocation. In contrast, when arrival rates increased to be more than the average service ($\lambda > \mu = 10$) TAGS performs worse than shortest queue. This is simply due to the fact that TAGS processes more jobs and in particular more long jobs, which by their nature have a longer response time. It is worth noting, the timeout values t for TAGS are the optimal values for the highest throughput as mentioned previously in Section 3 introduction. So, when optimising t for a different metric such as the average response time the result will be different Figure 8.

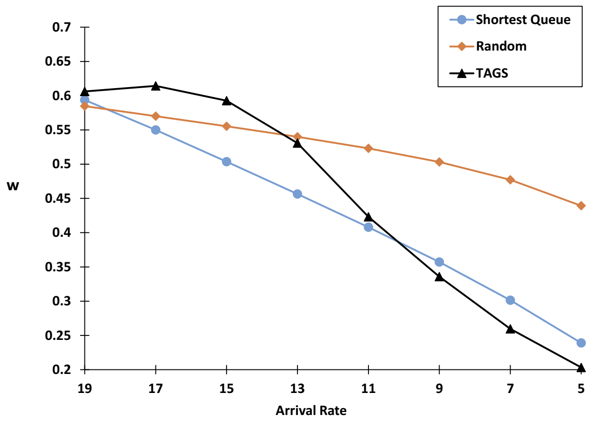


Fig. 8. Average response time varied against arrival rate λ, $\mu = 10$, $\alpha = 0.99$, μ_1=100μ_2

4 Energy model

The choice of the energy model is an integral part of the study that demonstrates how performance values convert into energy. We focus on the case where most of the server power consumption is due to the CPU, so we neglect other components such as memory and hard disk power consumption. The process we propose to estimate the energy consumed Ec by a server has an underlying assumption that this energy is essentially the processor performance states (P-states) multiplied by utilisation. To estimate the amount of energy per job, the throughput of the system is used, from which we can obtain an estimate of each job's cost in watts, along with utilisation and the P-state value.

The P-states are defined by the advanced configuration and power interface (ACPI) specification as the capability of a processor to switch between different supported operating frequencies. The P_0 state represents the highest performance state, which achieves maximum performance and consumes maximum power. States from P_1 to P_n are lower performance states, where n is the maximum P-state implemented by the processor, not to exceed 16. The higher P-state number refers to lower utilisation of the CPU and lower energy consumption. The number of the P-state is a processor-specific. For example, the AMD Opteron CPU has six performance levels with frequencies ranging from 1000 to 2600 MHz [19].

In this paper, the throughput, utilisation and P-state were considered in calculating total energy consumption and the average energy consumption per job. The system was assumed to be homogeneous, equipped with AMD Opteron CPUs. The value for each P-state in AMD Opteron CPUs was obtained from [2]. To specify the P-state, the utilisation of the system U at each arrival rate is calculated by equation 1.

$$U = 1 - Prob(Q_{Empty}) \qquad (1)$$

where $Prob(Q_{Empty})$ is the probability of the queue being empty.

Table 2 shows all P-state values. When obtaining the system utilisation and its related P-state value, the CPU total energy consumption can be calculated by

Table 2
P-state values

P-State	Power(W)	Clock (GHz)	Voltage (V))
P0	95	2600 MHz	140
P1	90	2400 MHz	135
P2	76	2200 MHz	130
P3	65	2000 MHz	125
P4	55	1800 MHz	120
P5	32	1000 MHz	110
Idle	15	-	-

equation 2.

$$Ec = \sum_{i=1}^{n}(SU_i \times SP_{i,active}) + ((1 - SU_i) \times SP_{i,idle}) \tag{2}$$

Wher SU_i is the $Server_i$ utilisation , $SP_{i,active}$ is the $Server_i$ P state value and $SP_{i,idle}$ is the $Server_i$ P state idle value.

$$AverageEnergyPerJob = Ec/T \tag{3}$$

The average energy consumption per job is calculated by equation 3, where Ec is the CPU total energy consumption and T is the system throughput.

4.1 Energy consumption of TAGS under exponential service demands

Results presented in this section, are discussed in terms of total energy consumption and energy per job under exponential demand. It is worth observing at this point that, the difference in total energy consumption between the shortest queue and random allocation insignificant under the exponential demand Table 3.

Table 3
Total Energy results and percentage difference

Arrival Rate	**19**	**17**	**15**	**13**	**11**	**9**	**7**	**5**
Total Energy _Random	192.433	184.659	174.863	163.712	151.898	139.852	127.749	115.638
Total Energy _Shortest	197.051	187.595	176.108	164.079	151.974	139.862	127.750	115.638
Total Energy _TAGS	205.775	202.326	199.299	191.352	178.001	161.647	144.743	127.777
% difference between Random and TAGS	6.7%	9.1%	13.1%	15.6%	15.8%	14.5%	12.5%	10.0%
% difference between Random and Shortest	2.4%	1.6%	0.7%	0.2%	0.0%	0.0%	0.0%	0.0%

Figure 9 shows the effect of varying the arrival rate on energy consumption by the TAGS algorithm, the shortest queue strategy and random allocation. Energy consumption is higher with TAGS than random allocation or the shortest queue in terms of total energy consumption under high and low arrival rates. The reason for this is related to the fact that the TAGS timeout mechanism assigns long jobs to the second node. The second server utilisation increases as it repeats the part of the

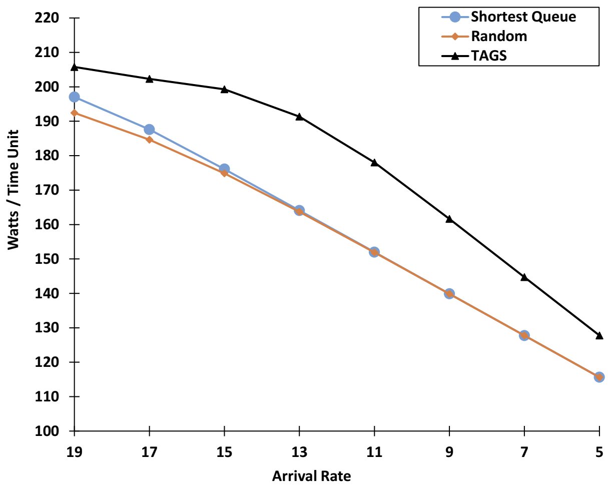


Fig. 9. Total Energy Consumption varied against arrival rate λ, μ= 10.

service that has been processed in the first node and then processes the subsequent part of the service. In total, the same job receives processing time equal to timeout at the first node plus the processing time at the second node. This behaviour excuses $server_1$ from long jobs, allowing more short jobs to be served, increasing the utilisation of $server_1$ as well. See Figure 5.

On the other hand, the shortest queue strategy assigns the job to the server with the shortest queue, and the job receives service until completion without interruption. Hence, the system utilisation at both nodes is balanced; there is no repeat, as the job is only being processed at one node.

Interestingly, while the difference in total energy consumption between random allocation and the shortest queue strategy is relatively small at each arrival rate, the difference between random allocation (the least energy consumption) and TAGS (the highest) is following a different trend. At the lowest arrival rate ($\lambda = 5$), the difference is approximately 10%, increases to be approximately 15.8% when the load increased to be 50% of the system capacity. Afterwards, the difference starts to decreases to be 6.7% at the highest arrival rate=19. This behaviour can be explained like that, TAGS mechanism from the beginning consuming more energy as the longer job receives processing in two servers with repeating of processing a part of the same job in both servers. In contrast, in the random allocation, the job receives the processing only once at one server, So the energy at low arrival rate by random is less than TAGS. However, when the load increased, the random allocation starts consuming more energy leading to a decrease in the difference in energy consumption between TAGS and random allocation.

In terms of average energy consumption per job, the TAGS algorithm with optimal timeout values at each arrival rate also consumes higher energy compared to the shortest strategy and random allocation. See Figure 10.

It is also worth pointing out that the TAGS algorithm can consume more energy per job when the arrival rate is relatively small. This can happen because the utilisation in both nodes at small arrival rates is less than the utilisation at higher

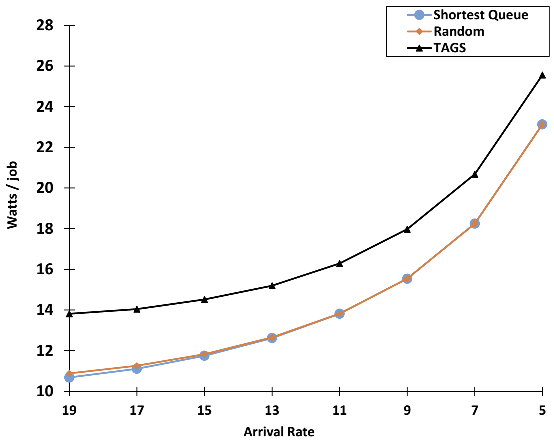


Fig. 10. Average Energy Consumption per job varied against arrival rate λ, μ= 10.

Table 4
P-state value at different arrival rates

	Utilisation		**Throughput**		**Energy Consumption**	
Rates	**5**	**7**	**5**	**7**	**5**	**7**
Server 1	31.96	44.74	3.29	4.47	62.03	69.77
Server 2	38.09	53.32	1.80	2.53	65.74	74.97
Throughput	-	-	5	7	-	-
TAGS Total	-	-	-	-	**127.78**	**144.47**
TAGS Energy per Job	-	-	-	-	**25.56**	**20.68**

arrival rates. Thus, the throughput is relatively small, but the energy reduction percentage is not as much as the reduction in the throughput percentage. This leads to an increase in the average energy per job. For example, when the arrival rate is five, the average energy consumption per job is more than when the arrival rate is seven. Table 4 illustrates the total energy consumption at rates five and seven, which is 127.78 and 144.47 watts per time unit, respectively, and the throughputs are 5 and 7 per time unit. The energy per job is 25.56 and 20.68 watts at arrival rates five and seven. Thus, while the increase in total energy consumption is approximately 13%, the throughput increased by 40%, which decreases the average energy consumption per job by 19.08%.

4.2 Energy consumption of TAGS under hyper-exponential service demands

We used the same methodology to calculate the energy consumption under hyper-exponential demand and exponential demand. In order to evaluate the TAGS energy consumption under the hyper-exponential distribution. we considered the scenario when the long job is 100 times longer than the short job $\mu_1 = 100\mu_2$, and the proportion of the short job is $\alpha = 0.99$. Figure 11 shows total energy consumption varied against arrival rate λ when service demand has a H_2 distribution. Results for energy consumption are shown for TAGS, shortest queue strategy and random allocation strategy.

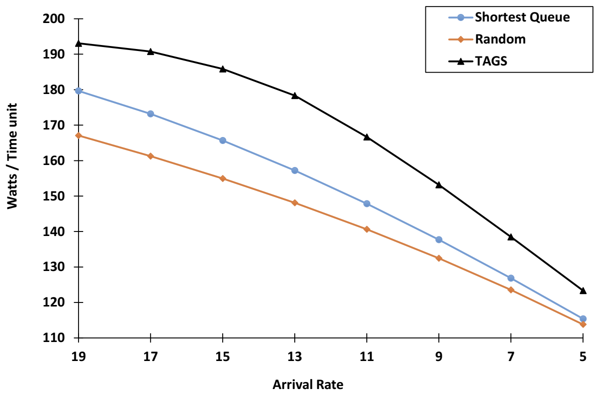
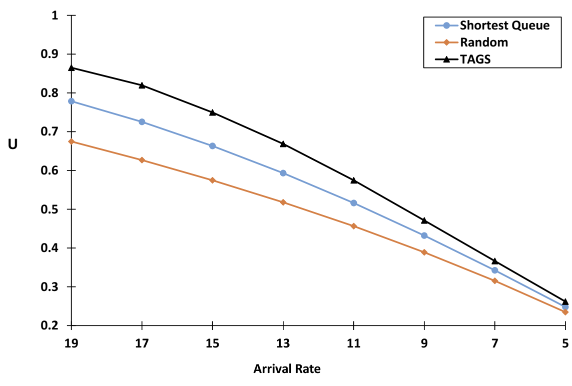


Fig. 11. Total energy consumption varied against arrival rate λ, $\mu = 10$, $\alpha = 0.99$, μ_1=100μ_2

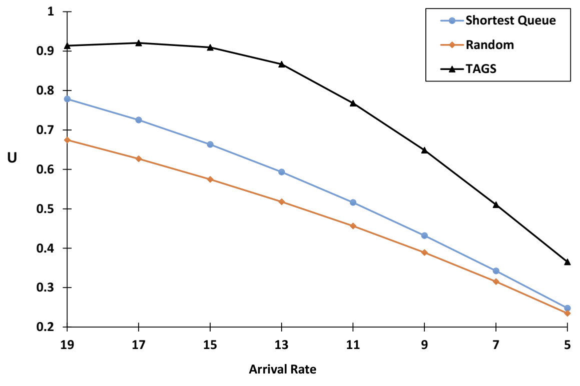
In Figure 11, we can see that TAGS consumed more energy than the shortest queue and random allocation at all arrival rates in terms of total energy consumption. It is also interesting to note that, while the difference in total energy consumption between TAGS and random allocation at higher arrival rate is as high as $\approx$ 14.44%, it is reduced to $\approx$ 8.4% at a low arrival rate λ=5. A possible explanation for this might be that at high arrival rate, the utilisation of $server_1$ and $server_2$ under TAGS is higher than the utilisation under the random allocation by 25% for $server_1$ and 30% for $server_2$, Figure 12. Both servers under the TAGS mechanism consume more energy than any server running under random allocation policy. While under the low arrival rate, the difference in the utilisation of $server_1$ between TAGS and the random allocation was reduced to $\approx$ 11% for $server_1$ but for $server_2$ the difference increased to 43%. It should also be noted that to calculate energy consumption we use the performance state (P value), which increases gradually when utilisation is increased. Hence, at high utilisation level, more energy is consumed.

In terms of energy consumption per job, TAGS consumes less energy than random allocation and the shortest queue when service demand exceeds 75% of the system capacity (arrival rate λ >15) Figure 13. We think the reason is related to the TAGS mechanism, which sends long jobs from the first server to the second server, clearing the way to serving more short jobs at the first server. That approach produces a higher throughput compared to random allocation strategies Figure 6. In contrast, at low arrival rate, TAGS consumes more energy per job than the shortest queue and random allocation. In fact TAGS performs poorly in energy consumption when the service demand is low as the TAGS mechanism involved repeating the amount of service from the timeout period at the second server.

The shortest queue is the optimal strategy in terms of energy per job under hyper exponential service demand as long as the service demand do not exceeds 75% of the system capacity.



(a) Server1



(b) Server2

Fig. 12. (a) $server_1$ utilisation and (b) $server_2$ utilisation, varied against arrival rate λ, $\mu = 10$, $\alpha = 0.99$, μ_1=100μ_2

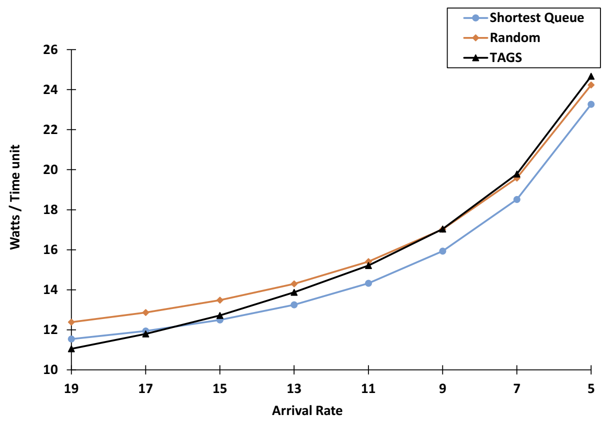


Fig. 13. Energy per job varied against arrival rate λ, $\mu = 10$, $\alpha = 0.99$, μ_1=100μ_2

All the three algorithms energy per job consumption shows an opposite trend to total energy consumption. While total energy consumption decreases at a low arrival rate, the energy per job increases. This behaviour is related to the fact that at a higher arrival rate, the utilisation increases, leading to more energy consumption, at the same time, the throughput of the system increases Figure 6. Consequently, the energy per job decreases at high arrival rates and increases at low arrival rates.

5 Conclusion and future work

We have studied energy consumption by the TAGS policy and compared it with that of the shortest queue strategy and random allocation. We have focused on these policies in the case of the high variability in workload. In our model, we assumed we have a two-node system where servers are identical regarding energy consumption and performance. The downside is that in practice, data centres are heterogeneous environments in which energy consumption may differ from server to server. We rely on the processor performance states (P-states) value to calculate energy consumption. The main downside of this is that we neglect energy consumption by other components of the server, such as hard disk and memory.

Our analysis of energy consumption under the exponential distribution and hyper-exponential distribution concluded that the TAGS mechanism consumes more energy than the other two policies regarding total energy consumption. The energy consumed per job followed the same trend under the exponential distribution. In contrast, under the hyper-exponential distribution, TAGS consumed less energy per job than random allocation when the arrival rate was high. The shortest queue was the optimal policy in that case.

In this paper, the primary focus was to evaluate and compare energy consumption by TAGS, shortest queue strategy and random allocation. No trade-off between energy consumption and performance metrics were introduced. It is vital to have a comparison mechanism between energy consumption and performance for each policy in order to have a clear decision regarding which policy should be chosen. We should develop a cost function that considers performance and energy in order to trade off energy and performance.

Variations among server specifications and processing capabilities are other factors that we should consider. We assumed servers are homogeneous regarding performance and energy while in reality, the data centre has a heterogeneous environment. It is worth while to study energy consumption when servers are not identical. If we have one server with low performance and high energy consumption and another server with high performance and low energy consumption, it will be valuable to determine which server should be the first server or second server. We are also aiming to investigate the benefit of using a multi-scheduling approach that combines TAGS with the shortest queue or random allocation to evaluate energy consumption.

References

[1] Osama Nasser Alrajeh and Nigel Thomas. Energy consumption of scheduling policies for htc jobs in the cloud. In *SimuTools*, pages 343–348, 2015.

[2] AMD. Power and colling in the data center. https://www.amd.com/Documents/34146A_PC_WP_en.pdf, 2005. [Online; accessed 01-October-2018].

[3] Anders SG Andrae and Tomas Edler. On global electricity usage of communication technology: trends to 2030. *Challenges*, 6(1):117–157, 2015.

[4] Jordi Arjona Aroca, Angelos Chatzipapas, Antonio Fernández Anta, and Vincenzo Mancuso. A measurement-based analysis of the energy consumption of data center servers. In *Proceedings of the 5th international conference on Future energy systems*, pages 63–74. ACM, 2014.

[5] Malith Jayasinghe BE. Task assignment in server farms under realistic workload conditions. 2011.

[6] Benjamin Berg, Jan-Pieter Dorsman, and Mor Harchol-Balter. Towards optimality in parallel scheduling. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):40, 2017.

[7] Pierre Delforge and Josh Whitney. Data center efficiency assessment-scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers. *Natural Resources Defense Council*, 2014.

[8] Cory Doctorow. Big data: welcome to the petacentre. *Nature News*, 455(7209):16–21, 2008.

[9] Dror G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, New York, NY, USA, 1st edition, 2015.

[10] Kristen Gardner, Mor Harchol-Balter, and Alan Scheller-Wolf. A better model for job redundancy: Decoupling server slowdown and job size. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on*, pages 1–10. IEEE, 2016.

[11] Suresh V Garimella, Tim Persoons, Justin Weibel, and Lian-Tuu Yeh. Technological drivers in data centers and telecom systems: Multiscale thermal, electrical, and energy management. *Applied energy*, 107:66–80, 2013.

[12] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, December 2008.

[13] Mor Harchol-Balter. Task assignment with unknown duration. *Journal of the ACM*, 49(2):260–288, March 2002.

[14] Jane Hillston. *A compositional approach to performance modelling*, volume 12. Cambridge University Press, 2005.

[15] Yichao Jin, Yonggang Wen, Qinghua Chen, and Zuqing Zhu. An empirical investigation of the impact of server virtualization on energy efficiency for green data center. *The Computer Journal*, 56(8):977–990, 2013.

[16] Jonathan G Koomey. Worldwide electricity used in data centers. *Environmental research letters*, 3(3):034008, 2008.

[17] Gary Shapiro. America's comeback starts with american innovators [soapbox]. *IEEE Consumer Electronics Magazine*, 1(1):19–24, 2012.

[18] Junaid Shuja, Kashif Bilal, Sajjad A Madani, Mazliza Othman, Rajiv Ranjan, Pavan Balaji, and Samee U Khan. Survey of techniques and architectures for designing energy-efficient data centers. *IEEE Systems Journal*, 10(2):507–519, 2014.

[19] George Terzopoulos and Helen D. Karatza. Power-aware bag-of-tasks scheduling on heterogeneous platforms. *Cluster Computing*, 19(2):615–631, 2016.

[20] Nigel Thomas. Comparing job allocation schemes where service demand is unknown. *Journal of Computer and System Sciences*, 74(6):1067–1081, 2008.

[21] Nigel Thomas and Jane Hillston. *Using Markovian process algebra to specify interactions in queueing systems*. University of Edinburgh, 1997.

[22] Mirco Tribastone, Adam Duguid, and Stephen Gilmore. The pepa eclipse plugin. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):28–33, 2009.

[23] Mueen Uddin, Jamshed Memon, Mohd Zaidi Abd Rozan, Raed Alsaqour, and Amjad Rehman. Virtualised load management algorithm to reduce co2 emissions in the data centre industry. *Int. J. Glob. Warm*, 7:3–20, 2015.