Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 318 (2015) 159–177

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Operating Policies for Energy Efficient Dynamic Server Allocation

Thai Ha Nguyen, Matthew Forshaw and Nigel Thomas

School of Computing Science, Newcastle University, UK

Abstract

Power inefficiency has become a major concern for large scale computing providers. In this paper, we consider the possibility of turning servers on and off to keep a balance between capacity and energy saving. While turning off servers could save power, it could also delay the response time of requests and therefore reduced the performance. Furthermore, as consistency is one of the most important factors for a system, we also analyse the level of consistency in the form of switching rate and fault occurrence. Several heuristic-based switching policies are introduced with a view to balance the cost between power saving, performance and consistency. Simulation results are presented and discussed with requests arriving according to a two-phase Poisson process.

Keywords: Energy efficiency, discrete event simulation, performance evaluation.

1 Introduction

The non-functional challenges facing large scale computing provision are generally well documented [13]. Amongst these the cost of energy has become of paramount concern. Energy costs now dominate IT infrastructure total cost of ownership (TCO), with data centre operators predicted to spend more on energy than hardware infrastructure in the next five years. The U.S. Environmental Protection Agency (EPA) attribute 1.5% of US electricity consumption to data centre computing [4], and Gartner estimate the ICT industry was responsible for 2% of global CO_2 emissions in 2007 [17]. With western european data centre power consumption estimated at 56 TWh/year in 2007 and projected to double by 2020 [3], the need to improve energy efficiency of IT operations is imperative.

Data centres, with their high density of power consumption and a steady growth in number, have become a major industrial energy consumer in the recent years. One of the most important factors that promoted their growth is that cloud computing

[1] Corresponding author: nigel.thomas@ncl.ac.uk

http://dx.doi.org/10.1016/j.entcs.2015.10.025

has become a big trend in web services and information processing. The most significant advantage of the cloud is its flexibility. It offers the chance of shifting capital expenditure to operational expenditure [7], which is ideal for starting a new service. Furthermore, since there is an increase in the quantities of data being collected for commercial, scientific or medical purpose, the big capacity of data centre is ideal to process such massive volume of data. As the cloud offers users an illusion of infinite computing resources on-demand [9], cloud computing is in fact essential to gather useful data from that enormous amount of information [1].

One of the more challenging problems in managing energy consumption in distributed systems is in handling variability of workload. There are a number of measures which can be applied to manage the effect of variable supply and demand. For example, there are a variety of load balancing techniques [11] and traffic shaping measures [12] which can be utilised to manage demand so that resources do not become excessively over-utilised when demand is high. An alternative approach is to dynamically manage the supply of service capability by making more servers availible during periods of high demand. Slegers *et al* [18,19] considered the problem of finding the optimal share of servers to different services under variable load in order to minimise a performance-based cost function.

This paper is based on the work of Slegers *et al* [20] and is focussed on the notion that servers can be powered off and on according to demand in order to avoid the non-trivial energy requirements of idle servers. With perfect knowledge of arriving workload an optimal dynamic allocation of servers can be obtained which significantly reduces the overall energy demand of the system with no impact on performance, i.e. servers can be made available only when they are going to be used. Of course, we do not generally have a perfect knowledge of future workload and so an optimal dynamic solution is not practical. Instead we must investigate the trade-off between energy consumption and performance (e.g. response time) to determine the best practical method of reducing energy costs whilst not adversely affecting the quality of service. Two principle approaches to minimising energy consumption are apparent. In the first instance an optimal fixed provision of servers can be computed based on estimated workload. Depending on the variability in demand, this approach might lead to servers being idle for extended periods or to some tasks experiencing long waiting times during peak demand. The second approach is to compute a strategy to turn servers on and off based on the current (or past) state of the system. This approach minimises idle time by turning off servers, but potentially delays tasks which arrive in a burst as it takes time to turn servers back on. In addition, powering servers off and on may lead to faults which not only reduce the total available number of servers, but may also further delay an arriving task.

The remainder of this paper is organised as follows. In the next section we explain the context of this work in relation to other work on energy reduction. In Section 3 we describe the system model and introduce six heuristic strategies for controlling the number of servers powered on and off. This is followed in Section 4 by a brief description of the simulation environmen and we then a present and discuss the results of our experiments. Finally we present some conclusions and directions of further work.

2 Related Work

Although the issue of power efficiency for data centres had received significant concerns in the recent years, much less attentions had been paid to the option of dynamically turning servers on or off depended on the incoming requests. Some researches that were close to the idea have been introduced, including [5] and [21]. However, they lacked the ability to dynamically turn servers on/off due to changes in the systems state. In [6], a dynamic server provisioning was examined using data traces from Windows Live Messenger. Since the workload of the Windows Live Messenger was periodic, i.e. it was predictable, this method was insufficient for the erratic and non-periodic workloads of a data centre. In [2], an M/M/c queueing system with removable servers and nontrivial setup times was examined. The queueing model was relatively similar to this project, but servers could only be powered up and down one at a time. Similar to [2], Gandhi *et al* [10] enabled multiple servers to be in powering mode at a time, while estimated the mean response time and the mean power consumption as key metrics.

Slegers *et al* [20,18] studied a model where the system consisted of a pool of homogeneous servers which could be in on, off or switching modes. The job requests arrived with high and low arrival rate and switching decisions were made by six different heuristics. The heuristics evaluated the number of jobs in queue along with the jobs arrival rate to optimize the system behaviour, given a trade-off between power consumption and response time. With a similar system architecture to [20], the work of Mitrani [16] focused on turning servers on and off in a block. The principle of this system was turning up a fixed number of reserves servers if the jobs queue exceeded a predefined threshold, while those reserves would be powered down in case of low workload. The two thresholds up (U) and down (D) determined the point of time when the system needed to turn on or off the reserves. In other words, m reserves servers would be powered up when the number of jobs exceed the threshold U, and those servers would be switched off when the jobs queue decreased to threshold D. Similar to [20], this system also used response time and power saving as key metrics. The switching question became how big the reserves should be, and how to choose the threshold U and D efficiently with the goal of balancing between performance and power consumption. Additionally, the paper also suggested the possibility of multiple server blocks with non-identical sizes. Server reserves could be powered up and down one after another as the job queue changed.

Yang *et al* [22] studied a resource management system which had the role of minimizing the number of active servers while keeping the overload probability to a standard threshold. This method did not measure the system performance by response time or energy consumption, but by an overload estimation model. The overload was calculated using the large deviation principle [8], while the decisions were made without any prior knowledge of the workload intensity. The dynamic cluster reconfiguration model which consisted of a resource management system and a batch of server nodes. The servers could be in either active or sleep modes, and heterogeneous servers cluster was supported. In the resource management system, a server scheduling strategy including server management and job scheduling was introduced. The task scheduler had the role to allocate suitable resources for requested tasks, while the cluster reconfiguration controller could dynamically turn servers on or off to satisfy the workload demand. In other words, the goal of the dynamic cluster configuration was to turn off as many servers as possible and still be able to comply with the quality-of-service constraints. Interestingly, this research also proposed the possibility of independently applying of techniques like DVS and DPM in individual servers to achieve fine-grained energy consumption control.

Noticeably, none of the works above had mentioned the factor of consistency in a system. The closest measurement of consistency for such problem was found in [15] in form of the rate of switching. That work focused on developing optimal policies for a single server system while taking the approximate response time, the energy consumption and the rate of switching into account. In this paper, we also consider consistency as the frequency of server powering on/off along with the occurrence rate of faults while switching. The measurement of consistency would be a beneficial metric for the policies to make switching decisions.

3 The Model

The system contained N homogeneous servers which can be in five states: powered up, powered down, powering up, powering down and faulty. The powered up servers could be working or staying idle, while there were only one mode each for the other states. The powered down mode was left ambiguous and could mean complete shutdown or hibernating, which consumed less or no power. During switching or fault modes, a server could not serve jobs but still consumed power. While faults could happen in almost any state, this paper only focused on the appearance of faults while switching, which was believed to have a high possibility of occurrence. Furthermore, the number of faults while switching should be an adequate index to measure the consistency of the system. Additionally, the modes were provided with specific costs, which were c_{up}, c_{idle}, c_{down}, c_{powUp}, $c_{powDown}$, c_{fault} respectively. While most of the costs denoted the relative energy cost for a server to run in that mode for a unit of time, c_{fault} reflected the relative loss for a server to remain in fault mode.

Furthermore, the system held an unbounded jobs queue which received job requests according to a two-phase Poisson process, i.e. a high and a low arrival period. During the periods, jobs were coming with the average mean of λ_{high} and λ_{low} for each unit of time correspondingly. The requests time was exponentially distributed with an average duration of μ, while the time of high and low periods were also exponentially distributed with mean ξ and η respectively. Similarly, other values were calculated using the exponential distribution, including the duration of fault t_{fault} and the switching time t_{up} and t_{down}. In addition from the energy costs and the fault cost, we also assigned the job holding cost c_{job}, which indicated the cost of holding a job in the system for one unit of time, in other words the need of processing jobs quickly. Moreover, there was also c_{pow} which reflected the energy saving benefit of keeping a server down for one unit of time. Again, those were only relative cost, i.e. $c_{pow} = 1$ and $c_{job} = 2$ simply meant that keeping a job in the system was double as expensive as the energy saving gained by powering down a server. To sum up, a data centres state could be described as follows:

$$S = \{j, \lambda, k_{up}, k_{down}, m_{up}, m_{down}, f\} \tag{1}$$

J denotes the number of jobs in queue, λ the arrival rate, while k_{up}, k_{down}, m_{up}, m_{down} and f are the number of servers which were currently up, down, powering up, powering down and at fault mode respectively. Furthermore, the sum of servers in all modes was always N (i.e there is no loss). A system state could move to the next state by the following possibilities:

- Staying the same without switching, the durations of modes and arrival periods can be decreased by one unit of time.
- If the duration of a server mode reaches zero, the number of servers for that mode will decrease by one and the number of servers up/down will be increased by one accordingly.
- If the duration of an arrival period reaches zero, the system will move to the next period.
- Turning on x servers, which means m_{up} will be increased by x and k_{down} will be decreased by x respectively.
- Turning off x servers, which means m_{down} will be increased by x and k_{up} will be decreased by x respectively.
- The number of jobs j will be increased according to the current arrival rate, meanwhile j can also be decreased if the duration of any job reaches zero.
- The same procedure is also true for f. It can be increased with the rate m_f as long as there are servers being powered on or off. If the duration of fault mode for a server reaches zero, f will be decreased.

In fact, those possibilities do not happen individually, they were often combined with others to reach the next state. Finally, there were two metrics of calculating the energy usage of the system. The first metric was the energy consumption which was the total of energy costs for all servers in the system. This was a straightforward method which was understandable and easy to calculate. However, it could not fully determine the efficiency of a data centre. For example, a system could keep the number of powered up servers low to gain small energy consumption, but it was in fact non-profitable since the inadequate number of servers could not keep up with the incoming requests. Therefore, the energy efficiency metric was introduced with the view of taking power consumption together with other values into account. This metric was calculated as follows:

$$E_{eff} = k_{down}c_{pow} - jc_{job} - fc_{fault} - m_{up}c_{powUp} - m_{down}c_{powDown} \tag{2}$$

In other words, this metrics calculated the power saving benefit of having a server staying down, while took into account the costs of faults, of having too many jobs in the queue and of switching too many servers. It was in fact a trade-off between saving powers, consistency and performance.

4 Switching Policies

This section will introduce six switching heuristics of different characteristics with the view of balancing data centres power savings, consistency and performance. The heuristics have the role to inspect each state of the data centre and made decisions of powering servers on or off for the next state. While most methods depended on statistical theories to achieve solution, some others simply reacted based on the number of requests in queue.

4.1 Static Allocation Heuristic

This method employs the concept of making no changes in the number of active servers. In other words, the heuristic decided on the best possible number of servers to switch on, and made no additional switching after that. Unless there were faults occurred in the process of powering on, then the heuristic would decide to switch on more servers to compensate for the lost ones. First, the heuristic determined the average rate for both arrival periods by adding the jobs loads and then divided it to the mean duration of both periods:

$$\alpha = \frac{\xi\lambda_{high} + \eta\lambda_{low}}{\xi + \eta} \tag{3}$$

Therefore, the problem became a jobs queue with arrival rate α and a fixed number of servers n, while n must not exceed the total number of servers N. This is a well-known problem in queueing theory (see e.g. [14]) and hence we can calculate the probability of all n servers being occupied. Subsequently the mean response time for n servers was determined as followed:

$$t_n = \mu\left(1 + \frac{Ec(n,\mu)}{n-\mu}\right) \tag{4}$$

As μ is the mean load of the system, $n-\mu$ indicates that n should not be smaller than the mean load. Furthermore, with the use of Erlang-C formula $Ec(n,\mu)$, the equation $\frac{Ec(n,\mu)}{(n\mu)}$ describes the extra percentage of time that the system may spend to finish processing the job in comparison with the average duration μ. In other words, it is the extra percentage of time that jobs need to wait before getting processed. Subsequently, it is trivial to calculate the energy efficiency of the data centre for that response time:

$$E_{eff} = (N-n)c_{pow} - c_{job}t_n \tag{5}$$

Since the method keeps a fixed number of servers all the time, the cost of switching and faults can be excluded. Therefore, the energy efficiency is only dependent on the power savings and the job holding cost. Then the process can be easily repeated for all possible number of servers in order to determine the best efficiency value and therefore the optimal number of servers.

The idea of the method is to sacrifice the ability to respond to incoming jobs volume for the stability. Obviously, this is an especially stable method. Since the heuristic only requires a small ammount of initial switching, it contains almost no faults. However, this method does not have a good performance as we will observe. In fact, the number of servers that this method decides on is often more than enough to handle the low arrival rate, while being insufficient for the high arrival rate. Therefore, in the case that the high arrival rate lasts longer than expected, this heuristic is likely to perform badly because of the high job holding cost.

4.2 Semi-static Allocation Heuristic

In order to fix the disadvantage of the static allocation heuristic, the semi-static policies is introduced. This method works with the same principle as the static allocation, but it treats the two arrival periods separately. In other words, the semi-static heuristic uses the Erlang C formula to find out the best number of servers for the high and low arrival periods separately. Therefore, those optimal numbers of serv-ers will be able to keep up with the arrival rates of both periods without the waste of turning on too many servers.

Unlike the static allocation method, the semi-static still needs to turn servers on or off between periods. However, since it uses the same mechanism as the static allocation, the semi-static does not take the switching time into account. Therefore, when the system is erratic, i.e. the durations of periods were short, or when the switching time is long, this method clearly shows a poor performance. In some extreme cases, the period might be over before the switching finished. Nevertheless, because switching times are often small in comparison to the periods duration and the length of jobs, the semi-static still has a good overall performance.

4.3 Idle Heuristic

This policy is the most straightforward method which depends entirely on the number of jobs in the queue. The idle heuristic has the strategy to turn off any idle server and turn on more servers if there are jobs waiting. To be more precise, the number of currently powering servers was also taken into account, i.e. more servers will be turned on if the number of jobs is larger than the total of active servers and powering on servers. This was clearly a very passive and unsophisticated method. While other heuristics try to predict the rate of incoming jobs and act correspondingly, the idle heuristic completely excludes the possibility of prediction. Understandably, this method requires a very high level of switching, which led to a big switching cost. Furthermore, even if the switching cost is small and the switching time is insignificant compared to the job duration, the idle heuristic would not necessarily be a good choice. If the job holding cost was significantly smaller than the power saving benefit, then it would be preferable to have some jobs waiting rather than turn on servers instantly.

Although the inefficiency and naivety of the idle heuristic is undeniable, its simplicity still possesses a strong point. While other methods would need a significant amount of computer resources to calculate statistical theories and run through various loops, the idle policy only requires a minimum of processing power.

4.4 Threshold Heuristic

This heuristic was proposes as an improvement from the idle heuristic. To be more precise, it is a generalisation version of the idle where a threshold j_{thres} is defined. If there were more than j_{thres} idle servers then the servers will be switched off. On the other hand, if the number of jobs is greater than the available servers and the threshold ($j > j_{thres} + k_{up} + m_{up}$) then more servers will be turned on. Clearly, the idle heuristic is equivalent to the case that $j_{thres} = 0$.

The idea of the threshold heuristic is to reduce the switching number of the idle heuristic by establishing j_{thres}. But the process of choosing a suitable threshold value is not trivial, especially when stability is also considered to measure the efficiency of a data centre. The threshold should significantly reduce the amount of switching, while not restricting the system too much to keep up with the incoming requests. Therefore, the task of selecting the threshold should take the job holding cost along with the power saving and the number of switching into account.

4.5 High/Low Heuristic

The high/low heuristic also depends on the current arrival period of the system. But unlike the semi-static, this method takes the switching time into account. The high/low can be considered the most sophisticated heuristic since it analyses every factor of the system, including the arrival periods, switching time, processing time and the queue length. Furthermore, we assume that the system is very stable. In other words, jobs will arrive at a constant state of λ_{high} or λ_{low} while the job duration had the fixed value of μ.

Basically, the high/low heuristic is based on the job processing speed to estimate the average time when the job is finished. If the system did no switching, then it is trivial to estimate the time that the jobs were finished. On the other hand, if the system decided to switch on more servers, then after the switching time, the processing speed would increase and the time would be shortened. On the contrary, the processing speed would be slowed down and the time would increase if the system switched off servers.

Assuming there was no switching and the current number of working servers is k_{up}, then the jobs are processed with the speed of $\frac{k_{up}}{\mu}$. If there are j jobs in the system and the arrival rate is λ, then the average decreasing rate of the queue length should be $s = \frac{k_{up}}{\mu} - \lambda$. Therefore, the finished time for those jobs is $t = j/s$. Since the queue is assumed to be processed at a constant rate, the estimated efficiency can be calculated as follows:

$$E_{eff} = K_{down} c_{pow} t - \frac{j}{2} c_{job} t \tag{6}$$

With similar approach, we can also calculate the efficiency when the system decides to switch servers on or off. The switching also includes the servers that are currently in powering up or powering down mode. In this case, the process can be divided into two different phases: before and after the switching time $t\prime$. Assuming at time $t\prime$ the number of jobs was reduced to $j\prime$ and the numbers of servers up and down were $k\prime_{up}$ and $k\prime_{down}$ accordingly. Then $j\prime$ can be calculated as $j\prime = j-(s/j_{up})$ or $j\prime = j-(s/t_{down})$ depending on the type of the switching. After that equation (6) can be used again to estimate the after-switching efficiency $E\prime_{eff}$ with $j\prime$ jobs and $k\prime_{up}$ working servers. On the other hand, the before-switching phase is estimated using the initial values. Then the total efficiency of the whole process should be:

$$E_{eff} = k_{down}c_{pow}t\prime - \frac{j+j\prime}{2}c_{job}t\prime + E\prime_{eff} - C_{switch} \tag{7}$$

Where C_{switch} is the switching cost, calculated as $(k_{up} - k\prime_{up})c_{powUp}$ for switching on and $(k\prime_{up} - k_{up})c_{powDown}$ for switching off. Subsequently, the process is repeated for every possible switching to choose an optimal outcome.

The high/low is a sophisticated heuristic which measures the performance along with the stability of the system. However, due to the complexity of the calculation, this method may require much more processing resource in comparison to others. Furthermore, as the high/low heuristic makes decisions based on the arrival period, it faces the same problem as the semi-static when the arrival periods were short and erratic.

4.6 Average Flow Heuristic

The average flow heuristic uses the same method of calculating the energy efficiency as the high/low heuristic. The only difference between them is instead of two types of arrival periods, this heuristic averages out the high and low arrival rate into a single one:

$$\lambda = \frac{\xi\lambda_{high} + \eta\lambda_{low}}{\xi + \eta} \tag{8}$$

The rest of the analysis is similar to equations (6) and (7). By having a single arrival rate, the average flow heuristic resolves the weakness of the high/low heuristic when the durations of periods were very short. This means the average flow is more stable and required fewer switchings. However, it cannot keep up with a long high arrival period as well as the high/low heuristic.

5 Experimental results

A simulation for a data centre model was implemented using Java JDK. It used an additional library of JFreeChart[2] to display the real-time running of the system, along with the statistical results. From this experiments were undertaken to better understand the performance of the various heuristics introduced above. Each simulation run lasted 10000 units of (simulated) time, and the costs were calculated from an average of 50 runs.

[2] http://www.jfree.org/jfreechart/

First, the heuristics were compared in a system with different levels of high arrival rate. Second, we simulated a scenario in which the ratios between the job holding cost and the cost for servers to staying off were varied. The results also indicated a significant improvement of the threshold heuristic over the idle heuristic. In addition, an erratic system with short duration of both high and low arrival rates explained the necessity of policies that are independent on what the current arrival period is. Furthermore, a scenario in which the server powering time was changed from low to high was also investigated. Finally, a chart of total faults among policies was displayed to explain why a stable heuristic is always preferable.

Note that the average costs in the figures below were the energy efficiency indexes of the data centres and not the energy consumptions. Those are only relative numbers that rate the performances and compares between heuristics, which means a negative value does not denote that there is no improvement in the data centre.

5.1 Increased arrival intensity

This experiment is one of the most common situations in practice when the duration of the high arrival period is relatively small in comparison with the duration of the low period. The system contained $N = 90$ servers with the average request processing time $\mu = 3$. The arrival rate in the low period was $\lambda_{low} = 10$, while the high arrival rate increased from 10 to 30 which made the utilization 100% at the highest peak. Durations of the low and high periods had a mean value of 100 and 10 respectively. The benefit of a server staying down was 1 while the job holding cost was 2. These relative values indicate that having a job in the queue was twice as expensive as having a server powered up. There was also a 0.05% rate of faults with the cost for each fault being 10, which slightly decreased the performance of the heuristics with high level of switching. Finally, the switching of servers up and down was considered with a cost of 3, while powering servers took 1 unit of time.

Figure 1 displays the performance of the heuristics with the setting of all-servers-on as a baseline cost, which clearly has the worst efficiency overall. However, as the high arrival rate kept increasing, the efficiency of the static allocation heuristic would eventually decrease to that of all-servers-on (since the static allocation will be all the servers). In certain occasions when the high period lasted longer than expected, the number of powered up servers would not be able to keep up with the increasing request, followed by an exceeding big jobs queue. On the other hand, the idle heuristic also performed badly in comparison with the rest policies. This was the most basic choice for a heuristic, but it involves too much switching which lleads to a high energy cost and higher occurrence of faults. As an improvement from the idle heuristic, the threshold heuristic with a threshold of 3, which was expected to have less switchings, showed a significantly better performance with its average costs always staying about 5 units beyond the cost of the former policy. On the other hand, the performance of the last three heuristics was quite remarkable with semi-static being the best policy because of its stability. It was also understandable that the semi-static and high/low heuristics, which made distinctive decisions for each arrival period, tended to perform better than other methods in the latter half

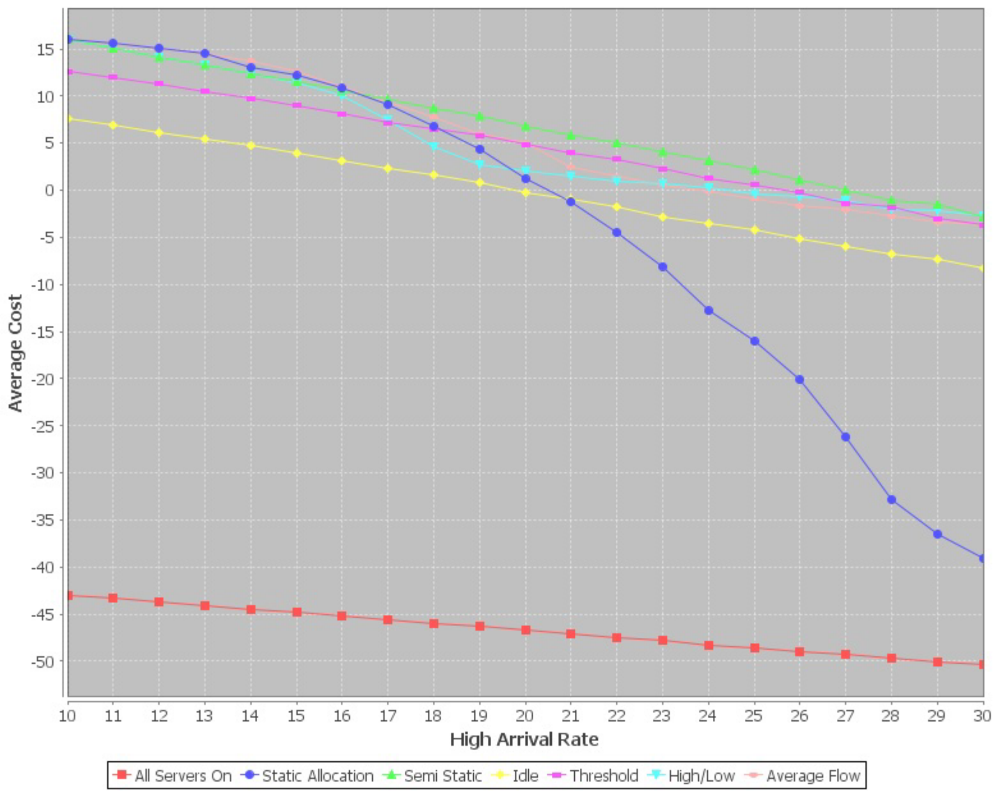

Fig. 1. The effect of increasing arrival intensity on heuristic performance

of the chart when the gap between arrival rates of high and low periods was bigger.

This scenario was one the most common situation in practice, when high periods only occurred at certain times of day. Out of the six heuristics, four performed quite well in this situation; however it was not enough to determine their performances and other characteristics would have to be taken into consideration in the later experiments.

5.2 Changing Cost Difference

While the last experiment focused on data centres whose priorities involved processing jobs quickly than having a server powered down to save energy, there are also systems which preferred to have their servers staying down until a certain level of jobs stacked up in the queue. Therefore, this section concentrates on the difference between the job holding cost and the benefit of servers powered off. Here the system contains 50 servers with mean request processing time $\mu = 3$. The other numbers were mostly the same as the last experiment, but the high and low arrival rates were 5 and 20, respectively. Additionally, the job holding cost had a value of 5, while the power saving of a server staying down was increased from 1 to 10, which indicated the priority of powering a server off in comparison with the need of quickly processing a job, from very low to very high priority.

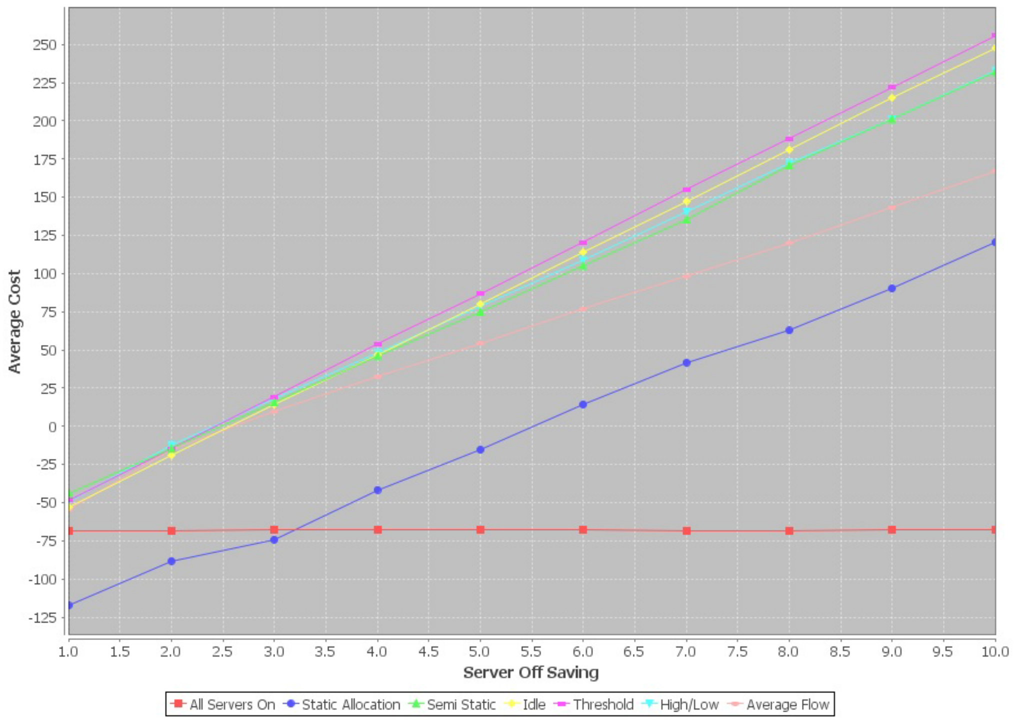

Fig. 2. The effect of increasing power cost on heuristic performance

As described in the previous section, the static allocation method performes worse when there is a big difference between high and low arrival rates. Except for the average flow policy, the other methods performed quite well with the threshold heuristic being slightly ahead of others. The idle heuristic also had a high result since its biggest disadvantage, the cost of powering servers on and off, is not significant enough in comparison with the job holding cost and the increased server off saving. On the other hand, the threshold heuristic with a threshold of 3 performs exceedingly well as the server off benefit grew. This method is well balanced between making servers stay down with its threshold and still keep up with the increasing requests. Apart from that, the average flow policy, while averaging out the high and low period, had a noticeable lower performance than others. Its behaviour was somehow similar to the static allocation heuristic when the disadvantage was caused by prolonged high arrival periods.

5.3 The Threshold Heuristic

Since the last two experiments showed a significant improvement of the threshold heuristic over the idle heuristic, this section considered those two separately from other methods to have a more clearly view of the pros and cons of the threshold method.

In this case a system with 50 servers was measured with a high arrival rate of 30 jobs per unit time and a low arrival rate of 5 jobs per unit time. The server off saving was 8 while the job holding cost was 10, which indicated that processing jobs quickly was given slightly more priority than saving power. The cost of powering servers on and off was accumulating from zero to 10 to show the changes of the two heuristics in comparison with each other. Furthermore, the threshold was set to be 5 to make the result easier to distinguish.

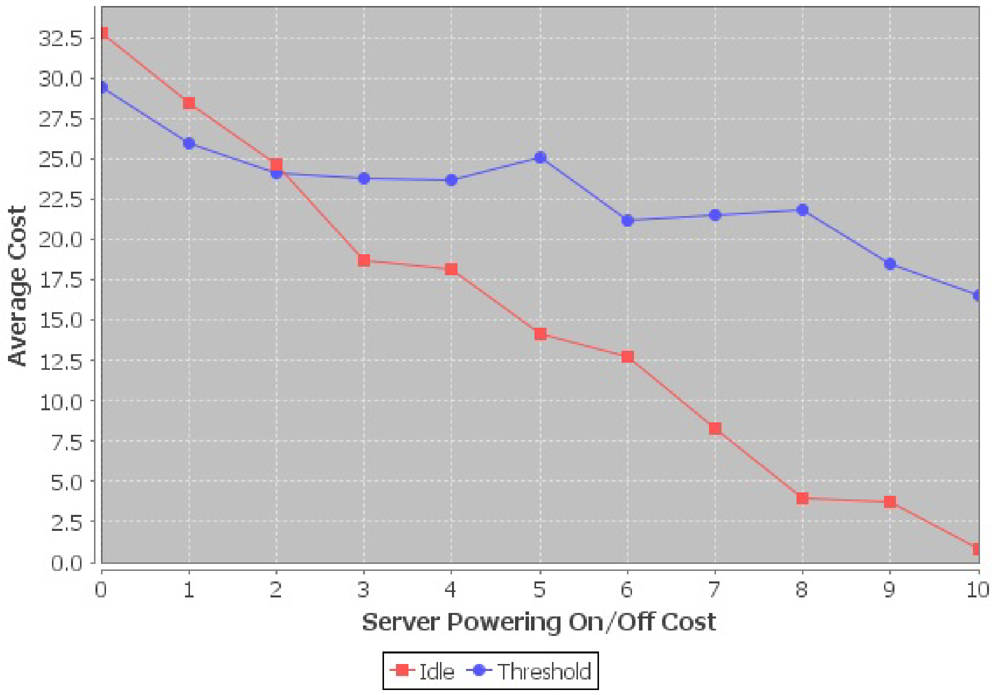

Fig. 3. The effect of increasing switching costs on threshold and idle heuristic performance

Not surprisingly, the idle heuristic had a better figure in the first part of the chart. Since the cost of turning servers on and off was notably lower than other costs, the advantage of threshold over idle heuristic was not significant enough. As the power-ing cost grown bigger, the threshold also showed its strong point and clearly surpassed the idle heuristic. It was also clear that the threshold was not always a good choice over the idle policy. For data centres which had a slow arrival rate of request but long mean processing time, servers would prefer to be turned on intermediately instead of waiting for the jobs queue to pass the threshold, which was likely to take a long time.

5.4 Changing Period Duration

In the previous scenarios, the high/low and semi-static heuristics displayed an impressive performance. The main common point of those two methods is that they both handled the high and low arrival periods separately. However, there are also systems in which those methods do not work that well. In the case when the durations of periods are very short, the erratic system may move to the next period before the decision of this period took effect, which may lead to unnecessary switching. Therefore, policies which calculate the average requests arrival might be more suitable. This experiment was designed to display such a case. The system was set up just like the first scenario with 90 servers with a high arrival rate of 25. The high arrival period lasted 10 units of time while the low arrival period was varied from 10 to 100 units of time, which moved from a very erratic data centre with short period durations to a more stable one.

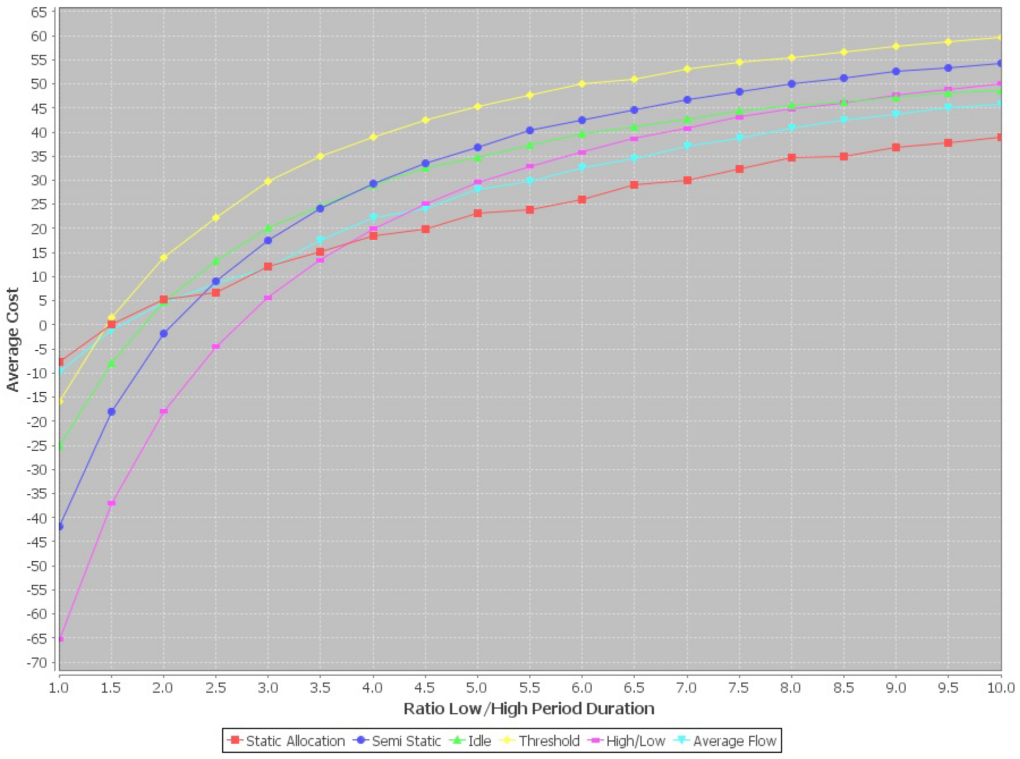

Fig. 4. The effect of increasing arrival duration on heuristic performance

The beginning of the chart displayed a considerable advance of the static allocation and average flow heuristics over the semi-static and high/low heuristics. As the first two methods made decisions based on the average flow of requests, which is less dependent on what period the data centre is currently in, they gain a big advantage when the duration of low and high periods are not too much different. This situation changed in the latter half of the chart when the low period duration increased and the two period-distinct policies regain their value. In addition, the naive policy of the idle heuristic also worked quite well when the system was erratic, while the threshold continued its good performance by being the best heuristic most of the time.

5.5 *Changing Switching Time*

In this scenario, the case of systems with different server powering time was investigated. This experiment was designed to simulate situations when a data centre need to handle small requests which require only a short processing duration, while turning servers on would require a longer time. The system was based on the data centre in the first experiment with 90 servers. The average service duration take 5 units of time, while the time of powering servers varied from 1 to 10.

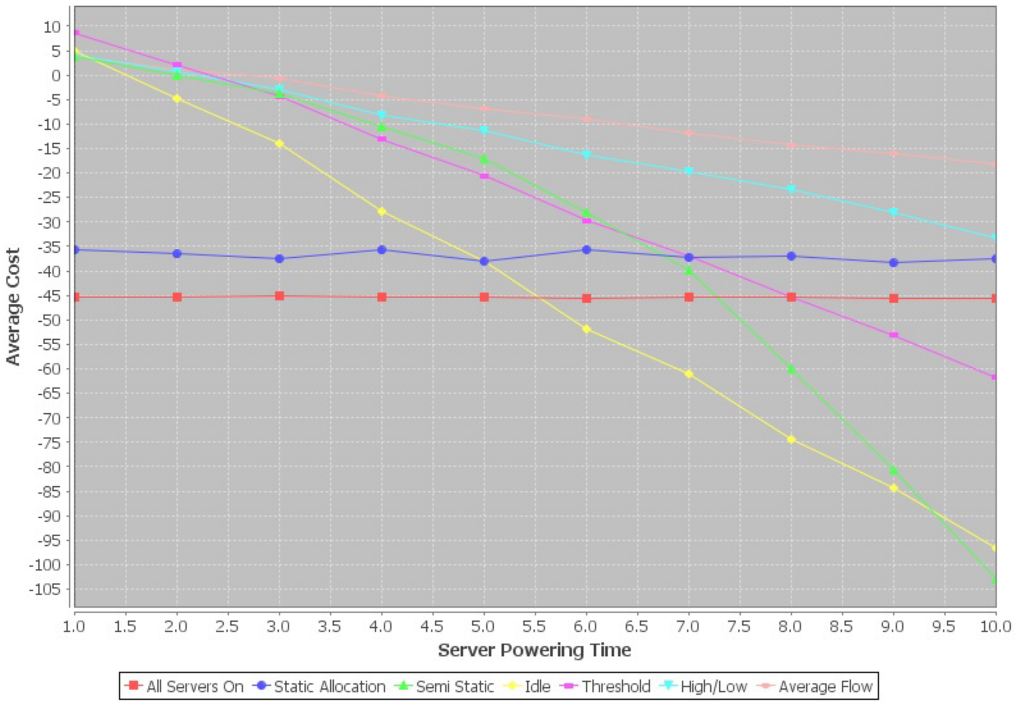

Fig. 5. The effect of increasing switching time on heuristic performance

As the server powering time increased, the performance from most heuristic gradually decreased, except for the static allocation method, since this heuristic did not need to turn on any more servers. The semi-static policy also made it decision without concerning about the powering time. Regardless of powering time, this method would behave the same. However, this heuristic still need to power servers on and off be-tween periods, which made semi-static the worst heuristic when the powering time passed 9 units. On the other hand, the idle and threshold policies decreasing patterns were quite similar with the threshold always performs better than its predecessor. As the switching time grew longer, more requests would have to wait before being pro-cessed. On the contrary, the high/low and average flow heuristic did take the switch-ing time into consideration, which gave them the best figures above all. However, as the switching time kept increasing, they would eventually be surpassed by the stabil-ity of the static allocation policy.

This scenario may not have much use in practice, when the switching time is often insignificant compared to the duration of requests. However, it did point out the prob-lem within semi-static heuristic that this method did not calculate the switching time. As semi-static being one of the most well performed heuristic in the last experiments, an enhancement regarding this problem would be really useful for further practices.

5.6 The Fault Rate

One of the most important factors when considering the performance of a system is its consistency. It is also true with data centres. In this case, the consistency denoted the ability to avoid unnecessary switching, which tended to trigger faults. The system in Section 6.1 was measured again to get the average number of faults for each heuristic after a loop of 10000 units of time and 50 runs, while the fault rate of switching was calculated with a probability of 0.1%.

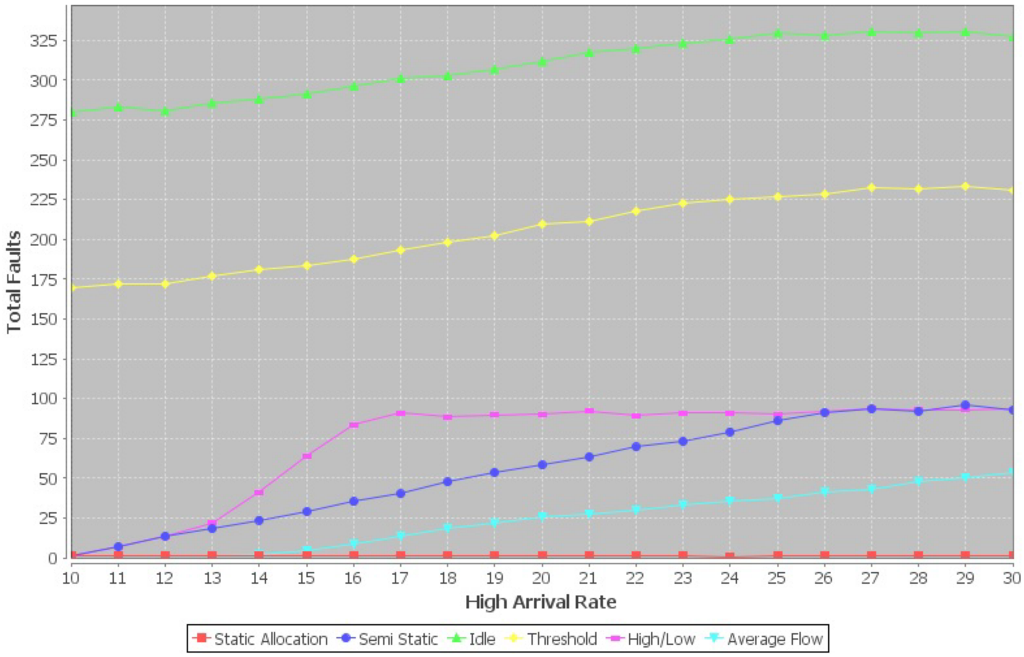

Fig. 6. The effect of increasing arrival intensity on fault rate

Obviously, the static allocation heuristic had the fewest faults above all, since this heuristic did not require any switching except for the initial powered on servers at the start of the system. On the other hand, it was understandable that the threshold and idle heuristics had the largest numbers of faults, as those two methods kept turning servers on and off to keep up with the length of the jobs queue. Moreover, the semi-static and high/low policies had quite a low level of faults, while the average flow method performed the most surprisingly with its number stayed as small as the static allocation method when the high arrival rate was low. The mean reason for the con-sistency of the average flow method was that this heuristic had a single average arrival rate, so its switching decision did not fluctuated between high and low periods like the last two heuristic.

As consistency being an indispensable factor for all systems, the number of switch-ing and faults will need to be taken into consideration when choosing a suitable heuristic for a data centre. The threshold policy, which had performed very well in the last experiments, may not be that impressive choice of a heuristic regarding its high number of switching.

5.7 Summary

From the experiments above, it is to be concluded that the idle heuristic is generally a poor choice of a policy. But it does not mean that this heuristic will perform badly in every situation. As an enhancement of the idle heuristic, the threshold heuristic showed some encouraging potential, however its high level of switching had become a great drawback. The static allocation method also did not perform well, since it preferred doing nothing over turning servers on, but that is also the reason why it has such high stability. The semi-static, high/low and average flow heuristics all have their own strengths and weaknesses which can be adapted for different situations. Especially in the case of the semi-static heuristic, if it could fix the problem of neglecting the switching time, this would be a really promising policy. Last but not least, as a stable system is always preferable, the consistency of the heuristics should be taken into account when considering their effectiveness.

There is clearly no best heuristic which suits every situation, as each heuris-tic works well in a specified situation and worse in others. There are many factors affecting the performances of the heuristic, including the differences between arrival rates and time, the length of switching time, the number of faults, etc. Since there is always a trade-off between performance and energy saving, it is the job of data cen-tres operators to find out the balance between them and to decide what policy is the most suitable for each specified data centre. Furthermore, there is even the possibility of having many policies for a single data centre, which can switch between different heuristics for different situation.

6 Conclusions

In this paper we have explored a model with multiple servers servicing an input stream of jobs. In order to limit the power consumption we allow servers to turn off and on according to demand. We have extended previous work in this area by considering the possibility that servers can fail whilst turning off or on. The costs of providing servers, holding jobs and failures have been incorporated into a new cost function which allows the performance of the system to be better understood.

We have proposed six heuristics for constructing a policy to manage the servers turning off and on and we have compared these numerically through a custom-built simulation. The simulation has been run with a number of scenarios to consider different operating conditions. The results of the simulation show that several of the heuristics are capable of performing well under certain conditions, but there is no single heuristic that we can claim is always best. This suggests that one line of future work might be to consider an environment which is capable of employing multiple heuristics to obtain a better performance under more conditions.

Our approach here has a number of limitations. Firstly we have only considered delays which are negative exponentially distributed, whereas in reality this may not be the case. Given that we are simulating the model, there is no real reason why we could not consider general distributions to better understand the effects that different distributions might have on system performance. We have also assumed that all servers are identical, whereas in practice this may not be the case. Not only would different servers have different processor speeds, but they would conceivably have different energy consumptions. It would be feasible and interesting to model more than one type of server and to consider, for example, the impact of utilising N fast but energy inefficient servers or M slower but more efficient ones.

References

[1] Armbrust, M., A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski and M. Zaharia, *A view of cloud computing.* Communications of the ACM, **53**(4), 50-58, 2010.

[2] Artalejo, J.R., A. Economou and M.J. Lopez-Herrero, *Analysis of a multiserver queue with setup times*, Queueing Systems, **51**(1-2), 53-76, 2005.

[3] Bertoldi, P. and B. Anatasiu., *Electricity Consumption and Efficiency Trends in European Union Status Report*, 2009.

[4] Brown, R., *Report to congress on server and data center energy efficiency: Public law 109-431*, Lawrence Berkeley National Laboratory, 2008.

[5] Chase, J.S., D.C. Anderson, P.N. Thakar, A.M. Vahdat and R.P. Doyle, *Managing energy and server resources in hosting centers*, ACM SIGOPS Operating Systems Review, **35**(5), pp. 103-116, 2001.

[6] Chen, G., W. He, J. Liu, S. Nath, L. Rigas, L. Xiao F. and Zhao, *Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services*, NSDI, **8**, pp. 337-350, 2008.

[7] Creeger, M., *Cloud Computing: An Overview*, ACM Queue, **7**(5), 2, 2009.

[8] Dembo, A. and O. Zeitouni, 'Large deviations techniques and applications (Vol. 2)', Springer, 1998.

[9] Fox, A., R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin and I. Stoica, *Above the Clouds: A Berkeley View of Cloud Computing*, University of California, Berkeley, Rep. UCB/EECS 28, 2009.

[10] Gandhi, A., M. Harchol-Balter and I. Adan, *Server farms with setup costs*, Performance Evaluation, **67**(11), 1123-1138, 2010.

[11] Gilly, K., C. Juiz, N. Thomas and R. Puigjaner, *Scalable QoS content-aware load balancing algorithm for a Web Switch based on classical policies*, in: 'Proceedings of the 22nd IEEE Internationall Conference on Advanced Information Networking and Applications', IEEE, 2008.

[12] Gilly, K., C. Juiz, N. Thomas and R. Puigjaner, *Adaptive Admission Control Algorithm in a QoS-aware Web System*, Journal of Information Sciences, **199**, 58-77, 2012.

[13] Jarvis, A., N. Thomas and A. van Moorsel, *Open issues in grid performability*, International Journal of Simulation, **5**(5), pp. 312, 2004.

[14] Kleinrock, L., 'Queueing systems, Volume I: Theory', 1975.

[15] Maccio, V.J. and D.G. Down, *On optimal policies for energy-aware servers*, in: 'Proceedings of the 21st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems', pp. 31-39, IEEE, 2013.

[16] Mitrani, I., *Managing performance and power consumption in a server farm*, Annals of Operations Research, **202**(1), 121-134, 2013.

[17] Pettey, C., *Gartner estimates ICT industry accounts for 2 percent of global CO_2 emissions*, 2007. http://www.gartner.com/newsroom/id/503867

[18] Slegers, J., N. Thomas and I. Mitrani, *Static and dynamic server allocation in systems with on/off sources*, Annals of Operations Research, **170**(1), 251-263, 2009.

[19] Slegers, J., N. Thomas and I. Mitrani, *Evaluating the optimal server allocation policy for clusters with on/off sources*, Performance Evaluation, **66**(8), 453-467, 2009.

[20] Slegers, J., N. Thomas and I. Mitrani, *Dynamic server allocation for power and performance*, in: 'Performance Evaluation: Metrics, Models and Benchmarks', pp. 247-261, LNCS 5119, Springer, 2008.

[21] Urgaonkar, B., P. Shenoy, A. Chandra, P. Goyal and T. Wood, *Agile dynamic provisioning of multi-tier internet applications*, ACM Transactions on Autonomous and Adaptive Systems, **3**(1), 1, 2008.

[22] Yang, J., K. Zeng, H. Hu and H. Xi, *Dynamic cluster reconfiguration for energy conservation in computation intensive service*, IEEE Transactions on Computers, **61**(10), 1401-1416, 2012.