Exact Analysis of Energy-Aware Multiserver Queueing Systems with Setup Times

Vincent J. Maccio
Department of Computing and Software
McMaster University
Hamilton, Ontario
Email: macciov@mcmaster.ca

Douglas G. Down
Department of Computing and Software
McMaster University
Hamilton, Ontario
Email downd@mcmaster.ca

***Abstract*—Energy consumption of today's datacenters is a constant concern from the standpoints of monetary and environmental costs. We model a datacenter as a queueing system, where each server can be switched on or off, with the time to switch a server on being nonnegligible. Previously derived structural properties of the optimal policy allow us to intelligently select policies to analyse further. Using the recursive renewal reward technique, we offer an exact analysis of these policies alongside offering insights, observations, and implications for how these systems behave. In particular, we provide insight into the question of the number of servers that should remain on at all times under a general cost function.**

I. Introduction

Immense energy consumption of datacenters has become a fact of modern life. The United States spends on the order of billions of dollars powering these systems each year [1], [2]. Google alone pays an annual energy bill on the order of hundreds of millions of dollars [3], [4]. While some may see this as an obligatory cost, the truth is many of these servers spend a significant amount of time idle. Moreover, an idling server uses a large percentage of the energy it would if it were busy [5]. To conserve costs, servers often have a lower energy state they can be switched to (off, sleep, etc.). However, the choice of if and when to make such a switch for each server is far from trivial. That is, while turning a server off *may* increase system efficiency, it *will* decrease system efficacy. This paper derives exact solutions which provide several insights into the behaviours of these systems, as well as answering key questions with regards to how they should be optimally provisioned and managed.

Due to the nature of these systems, queueing models are a natural analysis tool (for other, non-queueing theoretic approaches see [6]–[10]). To the best of our knowledge, Chen et al. [11] were the first to use queueing theory to tackle the problem of energy-aware provisioning in server farms. Around the same time Sledgers et al. [12] studied the problem with varying traffic rates where servers are allocated dynamically and presented heuristics to conserve energy. Since then, several variations on previously studied vacation models [13] have been developed, where vacations can be viewed as the setup time for a given server. Gandhi et al. began to study these systems in [14] and were able to present some interesting analytical results for the single server case, as well as some rules of thumb for the multiserver case. They continued their research in [15] in which they modelled a server farm as a continuous time Markov chain (CTMC) where servers begin setup if there is a job waiting to be served, and shut down as soon as they idle. As will be seen, employing a two dimensional CTMC model is a common and convenient way to view these systems. As such, in [16] Gandhi et al. introduced a method to derive moments of metrics associated with these CTMCs (such as the expected number of jobs in the system) called the recursive renewal reward (RRR) technique, where they also introduced another policy where servers wait some portion of time idle before being switched off. Phung-Duc [17] gave a comprehensive side by side comparison of RRR and other traditional methods for analysing these CTMCs. If the steady state of these CTMCs is also of interest, methods introduced by Doroudi et al. in [18] may be employed. Other authors have studied the same model as Gandhi et al. but under different policies (when servers turn on and off). Xu and Tian [19] studied the set of policies where e servers are turned off when there are d servers idle. Kuehn and Mashaly [20] analysed policies which wait for a threshold of jobs to accumulate in the queue before a server starts its setup and turns servers off when they idle, under the presence of a finite buffer. Lastly, Ren et al. [21] analysed a finite two-dimensional CTMC similar to Kuehn and Mashaly in the context of virtual networks, which allows for a number of servers to always remain operational, but omits the use of turn on thresholds.

Limiting study to the single server case grants an even greater understanding of these systems. Artalejo [22] was one of the first to look at this case under general processing time distributions. However, his work focused on particular vacation models which do not fully capture the behaviour of a server which can be switched on and off. In [23] we adapted these models to better suit the domain of green computing, and were able to derive the optimal policy for the single server case under complete generality with regards to the underlying distributions and cost function. Gebrehiwot et al. [24] extended the analysis of the single server case by allowing multiple sleep states, and more recently looked at the model under the *processor sharing* service discipline [25]. Hyytiä et al. [26], [27] also studied this model under processor sharing in addition to *last come first serve*, and different routing configurations.

While the contributions of the previously mentioned works are substantial, a gap in knowledge still remains. While optimal control of the single server case is well understood, it offers less practical application than corresponding multi-server models. However, when studying the multiserver case complexity constrains researchers to focus on specific policies, which in general may be far from optimal. Therefore, we studied the structure of the optimal policy in [28] and derived several structural properties which greatly aid in policy selection. In this paper we leverage our past work to intelligently select policies to analyse, as well as drawing key conclusions regarding the optimal policy. The main contributions of this work are as follows.

1) The description and exact analysis of two distinct policies, *bulk setup* and *staggered threshold*.
2) A range of numerical experiments which yield exact values for metrics of interest.
3) An examination and discussion of these numerical results which lead to several insights into how these systems behave, specifically with respect to the question of the number of servers one should always leave on.

For further details and discussion on the results presented here, we direct the reader to the corresponding technical report [29].

II. Model

We analyse a system with C homogeneous servers and a central queue. Jobs arrive to the system following a Poisson process with rate λ, are processed on a first come first served basis, and have processing times which are exponentially distributed with rate μ. Each of the C servers can be in one of four energy states, *off, setup, idle*, or *busy*. For ease of exposition we often refer to a server being busy, idle, off, or in setup as shorthand for a server being in the corresponding energy state. Regarding definitions and transitions, a server is *idle* if and only if it is on and not processing a job. Furthermore, a server can only begin serving a job if it is currently *idle*, in which case the server becomes *busy*. At any time a server can be instantly switched *off*. Furthermore, an *off* server can transition to *setup*. This is often referred to as a server starting to turn on. Once in *setup*, the server will remain there for a time exponentially distributed with rate γ, after which the server will become *idle*. In other words, each server has setup times expected to last $1/\gamma$ time units, while turn-offs happen instantaneously.

The nature of the policies considered in this work, alongside the assumptions on the underlying distributions for the arrival, processing, and setup times, allow the system to be modelled as a CTMC. The corresponding state space of the CTMC is denoted by (i, j), where i is the number of servers currently on (*idle* or *busy*), and j is the number of jobs currently in the system (including those in service). For such a CTMC, one can impose a policy which determines the transition rates between these states. For the policies described in this work, we separate the C servers into one of two groups, static or dynamic. That is, C^* of the C servers will be static (always remain on) and $C - C^*$ servers will be dynamic (can be switched on or off), where $0 \leq C^* \leq C$. For our purposes, C^* is treated as a decision variable for each policy. Furthermore, a policy is fully described when the setup and turn off criteria of each of the remaining $C - C^*$ dynamic servers is given. A graphical representation of this model can be seen in Figure 1.

It is worth noting that in future sections there is often a threshold value associated with turning servers on, denoted by k. This threshold value k is also viewed as a decision variable. Due to the Markovian nature of the model, optimal choices for switching servers on/off are always made the moment the system enters a state (the moment an event occurs). These decision variables are left abstract, and their allowable range is determined by the employed policy. For example, for a system with $C = 2$, it may be the case that for all states $(1, j)$, where $j > 3$, the second server will begin its setup, if it has not yet done so. Furthermore, for the same system, it may be the case that for all states $(2, j)$ where $j < 3$, the second server is immediately switched off.

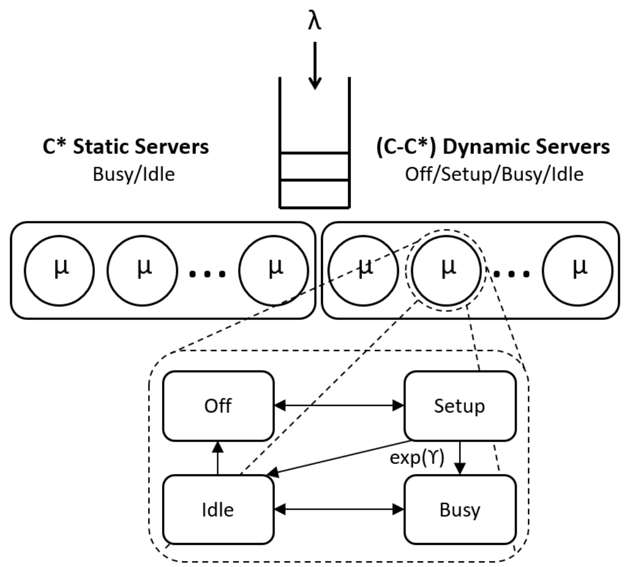


Fig. 1: The model under study. Dynamic servers take time exponentially distributed with rate γ to move from *setup* to *idle* or *busy*, all other transitions happen instantly if the system state allows it.

A. Metrics and Notation

In order for one to compare policies there must be some associated metrics with which to make comparisons. In this work we focus on the trade-off between two metrics. To measure efficacy we examine the expected response time, denoted by $\mathbb{E}[R]$. To measure efficiency, we examine the expected excess energy cost (subsequently referred to as expected energy cost or expected rate of energy consumption) denoted by $\mathbb{E}[E]$. Without loss of generality, the expected energy cost is the sum of the expected number of idle servers and the expected number of servers in setup, each weighted by some factor normalized to how much energy they use compared to a busy

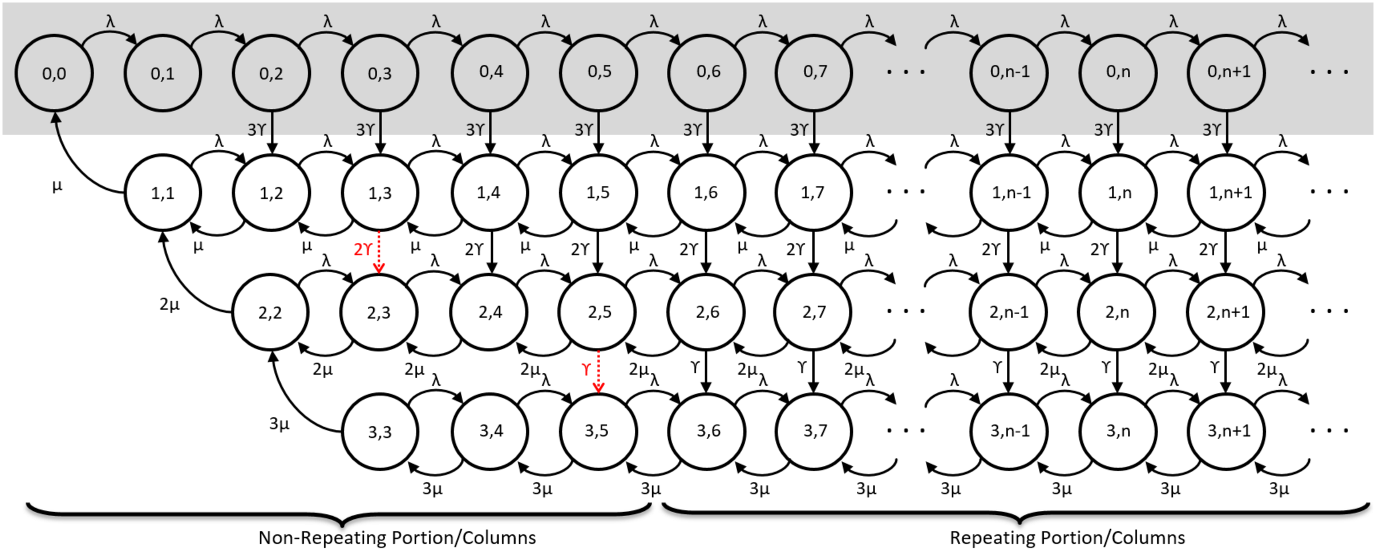


Fig. 2: Bulk setup CTMC with $C^* = 0$ and $k = 2$. If C^* is changed from 0 to 1, the shaded row would be merged into row 1 creating the state $(1, 0)$ as well as adding the dashed red arrow transitions to states $(1, 3)$ and $(2, 5)$. Moreover, the repeating portion of the CTMC would now start at column five.

server. That is, a server in setup accumulates energy cost at rate denoted by $r_{setup} = 1$, while an idle server accumulates it at some factor less than r_{setup}, denoted by r_{idle}, where $0 < r_{idle} < 1$. While it may seem odd that we disregard energy costs associated with processing jobs, there is good reason to do so. For any stable system, all jobs which enter will have to be processed eventually. Therefore, in steady state the energy cost accumulated by processing jobs is completely insensitive to which policy is chosen.

Due to the structure of these CTMCs, they can be analysed using the RRR method described in [16], which allows for the exact analysis of the expected rate of energy consumption, and the expected response time of the system. The idea of the method is to build recursions for costs based on how much of said cost is incurred before transitioning one column left of a given state. Specifically, if the system currently contains j jobs, one must keep track of how much of a particular cost is incurred before the system contains $j - 1$ jobs. For our purposes, the costs we derive are the expected amount of time, the expected holding costs, and the expected total energy costs incurred before transitioning one column left. For state (i, j) we denote these values by $T_{i,j}$, $H_{i,j}$ and $E_{i,j}$, respectively. As a visual aid, in Figure 2, $T_{1,3}$ would denote the expected amount of time for the system to reach one of the states $(0, 2)$, $(1, 2)$, or $(2, 2)$, given that it started in state $(1, 3)$. The value $H_{0,5}$ would denote the expected amount of holding cost incurred during the time the system transitions from state $(0, 5)$ to one of the states $(0, 4)$, $(1, 4)$, $(2, 4)$, or $(3, 4)$. Furthermore, to build a recursive relationship between all of these values, one must know the probability of being in a particular state once a left transition has been made. Therefore, we denote the probability of being in row i' after moving one column left of state (i, j) by $P_{i'}(i, j)$. In Figure 2, $P_2(0, 4)$ would denote the probability of being in state $(2, 3)$ the moment the system reaches one of the states $(0, 3)$, $(1, 3)$, $(2, 3)$, or $(3, 3)$, given it started in state $(0, 4)$. The recursions for these costs and probabilities are "tied off" once the CTMC reaches the *repeating portion*. Informally, the repeating portion of the CTMC is when the states in any column to the right of the current column are indistinguishable from the corresponding state in the current column, based on the transition rates alone. The non-repeating portion of the CTMC consists of the states belonging to columns in said CTMC before it starts repeating. Again, using Figure 2 as a visual reference, the repeating portion starts with column 6, and continues right to infinity. This is because states $(2, i)$, where $i \geq 6$ move to state $(3, i)$ with rate γ, while state $(2, 5)$ cannot move directly to state $(3, 5)$.

III. Analysis

Here we analyse two distinct policies, *bulk setup* and *staggered threshold*, with the ultimate goal of deriving the expected rate of energy consumption ($\mathbb{E}[E]$), and the expected response time ($\mathbb{E}[R]$). With the notation introduced in the previous section, these expressions may firstly be written down independent of which policy is being employed. That is, from the renewal reward theorem we know that the expected number of jobs in the system ($\mathbb{E}[N]$) is the expected holding cost incurred over a renewal cycle, divided by the expected time to complete that same renewal cycle. For simplicity we choose the reference state for this cycle to be the state $(C^*, 0)$, i.e. when the system is empty. From the renewal reward theorem and Little's law we can write:

$$\mathbb{E}[R] = \frac{\mathbb{E}[N]}{\lambda} = \frac{H_{C^*,1}}{\lambda(T_{C^*,1} + 1/\lambda)}. \tag{1}$$

We can write a similar expression for $\mathbb{E}[E]$,

$$\mathbb{E}[E] = \frac{E_{C^*,1} + (r_{idle}C^*)/\lambda}{T_{C^*,1} + 1/\lambda}. \quad (2)$$

It is also noted that the underlying CTMCs of all threshold policies, which includes all previously well-studied policies, have identical repeating portions. Moreover, the optimal policy is known to be a threshold policy [28]. Therefore, we can derive all values associated with the repeating portion independent from the choice of policy. These values are as follows,

$$0 = \frac{\lambda}{\lambda + i\mu + (C-i)\gamma} P_i^2(i) - P_i(i) + \frac{i\mu}{\lambda + i\mu + (C-i)\gamma}$$

$$P_{i'}(i) = \frac{(C-i)\gamma P_{i'}(i+1) + \lambda \sum_{m=i+1}^{i'-1} P_{i'}(m) P_m(i)}{i\mu + (C-i)\gamma + \lambda(1 - P_i(i) - P_{i'}(i'))}$$

$$T_i = \frac{1 + (C-i)\gamma T_{i+1} + \lambda \sum_{m=i+1}^{C} T_m P_m(i)}{i\mu + (C-i)\gamma - \lambda P_i(i)}$$

$$H_{i,j} = \frac{j + (N-i)\gamma H_{i+1,j} + \lambda(T_i + \sum_{m=i+1}^{C} H_{m,j} P_m(i))}{i\mu + (N-i)\gamma - \lambda P_i(i)}$$

$$E_i = \frac{(C-i) r_{setup} + (N-i)\gamma E_{i+1} + \lambda \sum_{m=i+1}^{C} E_m P_m(i)}{i\mu + (C-i)\gamma - \lambda P_i(i)}$$

where $i' > i$, and $P_{i'}(i)$, T_i, and E_i are used as shorthand for $P_{i'}(i,j)$, $T_{i,j}$, and $E_{i,j}$ respectively, due to their independence from j in the repeating portion of the CTMC. While it is true that even in the repeating portion of the CTMC $H_{i,j}$ is dependent on j, one can still obtain a closed form expression after the observation that $H_{i,j+1} = H_{i,j} + T_{i,j}$.

A. Bulk Setup

With all common derivations out of the way, we proceed with our examination of specific policies. The first of the two policies we analyse is the bulk setup policy. Firstly, this policy has C^* static servers, where C^* is treated as the first of two decision variables. The second decision variable is the threshold value k. If there are ever $C^* + (d+1)k$ or more jobs in the system ($C^* + (d+1)k$ or more jobs waiting in queue or being served), where d is the number of dynamic servers currently on, the system will immediately start the setup process for all remaining dynamic servers, hence the name. When there are less than $C^* + (d+1)k$ jobs in the system, all servers in setup will be immediately switched off. Furthermore, a dynamic server which has completed its setup process and is now on will be switched off the moment it idles (not when the number of jobs drops below the setup threshold). While this policy may seem needlessly aggressive and is admittedly unappealing from an implementation standpoint, the mass setup nature of the policy has been shown to be optimal for linear cost functions [28]. Moreover, the analysis of this policy grants insights into the overall behaviour of these systems. A graphical representation of an underlying CTMC employing the bulk setup policy can be seen in Figure 2, as well as how the Markov chain would be altered by changing C^*.

For the sake of clarification, we give a detailed derivation of the term $T_{i,j}$ under this policy. While the derivation of other terms is not exactly the same, it should be similar enough for the reader to see the general form and technique. However, if more detail is required, we direct the reader to [29] where all derivations are given in full. Firstly consider the case where $j < (i - C^* + 1)$, i.e. when there are no servers currently in setup. The expected amount of time to transition one column left of state (i,j) is broken down into a sum of the expected amount of time for the next event to occur, and the probability of each event occurring next multiplied with the expected amount of time to reach column $j-1$ from the new system state. This point is more easily described through the use of Figure 2. Consider the case of $T_{2,3}$ corresponding to state $(2,3)$ in Figure 2. This is in fact a simple case since due to the direction of the arrows it is known that when arriving at column 2 for the first time after leaving state $(2,3)$, the system must be in state $(2,2)$. Therefore, we can view $T_{2,3}$ as the expected amount of time until the system reaches state $(2,2)$ from state $(2,3)$. This is the expected amount of time to leave state $(2,3)$ plus some other term(s). Concerning the next event witnessed, the only two possibilities are an arrival or a departure. After a quick observation however, one will note that if the next event is a departure, then the system is in state $(2,2)$ and no more time will be added. Therefore, this case may be excluded from the expression. This leaves the case of an arrival. When a arrival is the next event, the system moves to state $(2,4)$. Therefore, we must now derive the expected amount of time to move from state $(2,4)$ to state $(2,2)$. At first glance this may appear to be daunting as there are an infinite number of paths the system may take before transitioning to state $(2,2)$, but this value may be abstracted and expressed using our previously defined notation. That is, now we are interested in the expected amount of time it takes to move two columns left of state $(2,4)$. With this observation we can now write the following expression,

$$T_{2,3} = \frac{1}{\lambda + 2\mu} + \frac{\lambda(T_{2,4} + P_2(2,4)T_{2,3} + P_3(2,4)T_{3,3})}{\lambda + 2\mu},$$

and after some algebra,

$$T_{2,3} = \frac{1 + \lambda T_{2,4} + P_3(2,4)T_{3,3}}{2\mu + \lambda(1 - P_2(2,4))}.$$

This line of thinking can naturally be extended to the general case to write an expression for $T_{i,j}$, recall we are currently assuming $j < (i - C^* + 1)$:

$$T_{i,j} = \frac{1 + \lambda(T_{i,j+1} + \sum_{m=i}^{C} T_{m,j} P_m(i,j+1))}{\lambda + \min(i,j)\mu}.$$

All that is required to derive an expression for the case where $j \geq (i - C^* + 1)$ is to account for the possibility that the next event to occur could now be a setup completion. Again, this can naturally be extended to the following expression,

$$T_{i,j} = \frac{1 + (C-i)\gamma T_{i+1,j} + \lambda T_{i,j+1}}{\lambda + (C-i)\gamma + \min(i,j)\mu} + \frac{\lambda \sum_{m=i}^{C} T_{m,j} P_m(i,j+1)}{\lambda + (C-i)\gamma + \min(i,j)\mu}.$$

Rearranged versions of the expressions for $T_{i,j}$, where $T_{i,j}$ is isolated, are given later in this section. For now we focus on how one would arrive at a value for these *time* values. At first look, it seems at best one would have to solve a system of equations, where there is an equation for each state in the non-repeating portion. This is actually not the case as the CTMC has structure which can be exploited. For example, inspecting $T_{2,5}$ and expanding the expression, one can note it is only dependent on the other expected transition time values $T_{2,6}$, $T_{3,6}$, and $T_{3,5}$. This is due to the fact that $T_{2,6}$ and $T_{3,6}$ lie in the repeating portion of the CTMC, and therefore have associated closed form expressions, and furthermore $T_{3,5}$ is only itself dependent on $T_{3,6}$. In fact, if the order in which the expected transition times are solved is chosen intelligently, the complexity of solving these values can be drastically reduced from solving each value simultaneously as a system of linear equations. This is done by visually noting that the transition times are dependent only on the corresponding transition times to the states below, and to the right of them. Mathematically, if $i' < i$ or $j' < j$, then $T_{i,j}$ is not dependent on $T_{i',j'}$. That is, the correct order to solve these values is to start with the state in the bottom right corner of the non-repeating portion, state $(3,5)$ in Figure 2, state $(C, (C-C^*+1)k + C^* - 1)$ in the general case (assuming $C^* < C$), and then begin moving to the left, solving the corresponding values for each state until the end of the row (state (C,C)) is reached. At this point, the procedure would move one row down, and again begin moving left until the end of the row is reached, solving the corresponding values along the way. The non-repeating portion of the chain is iteratively traversed in this way until all states are exhausted. Noting that there are approximately $(C-C^*)C^*(k+1)$ states in the non-repeating portion, such a recursion solves all $T_{i,j}$ values with complexity $O((C-C^*)^2 C^* k)$, as opposed to the complexity of simultaneously solving the system of equations, which is $O(((C-C^*)C^*k)^3)$.

With the typical approach and procedure explained, we give all recursions and information needed to evaluate equations (1) and (2). Firstly, due to the servers turning off when idle, the following boundary conditions are known: $(\forall i > C^* : \; P_{i-1}(i,i) = P_{i-1}(i-1,i) = 1)$ and $(\forall j \le C^* : \; P_{C^*}(C^*, j) = 1)$. Secondly, we present all expressions pertaining to the non-repeating portion of the chain where no servers are in setup, i.e. when $j < (i - C^* + 1)k + C^*$, $C^* \le i \le C$, and $i \le i'$. They are as follows:

$$P_i(i,j) = \frac{\min(i,j)\mu}{\min(i,j)\mu + \lambda(1 - P_i(i,j+1))}$$

$$P_{i'}(i,j) = \frac{\lambda \sum_{m=i+1}^{i'} P_{i'}(m,j) P_m(i,j+1)}{\min(i,j)\mu + \lambda(1 - P_i(i,j+1))}$$

$$T_{i,j} = \frac{1 + \lambda(T_{i,j+1} + \sum_{m=i+1}^{C} T_{m,j} P_m(i,j+1))}{\min(i,j)\mu + \lambda(1 - P_i(i,j+1))}$$

$$H_{i,j} = \frac{j + \lambda(H_{i,j+1} + \sum_{m=i+1}^{C} H_{m,j} P_m(i,j+1))}{\min(i,j)\mu + \lambda(1 - P_i(i,j+1))}$$

$$E_{i,j} = \frac{\max(i-j,0) r_{idle} + \lambda E_{i,j+1}}{\min(i,j)\mu + \lambda(1 - P_i(i,j+1))} + \frac{\lambda \sum_{m=i+1}^{C} E_{m,j} P_m(i,j+1)}{\min(i,j)\mu + \lambda(1 - P_i(i,j+1))}.$$

Lastly, the expressions for the non-repeating portion of the chain where servers are in setup, i.e. when $j \ge (i - C^* + 1)k + C^*$, $C^* \le i \le C$, and $i \le i'$, are as follows:

$$P_i(i,j) = \frac{\min(i,j)\mu}{\min(i,j)\mu + (C-i)\gamma + \lambda(1 - P_i(i,j+1))}$$

$$P_{i'}(i,j) = \frac{(C-i)\gamma P_{i'}(i+1,j)}{\min(i,j)\mu + (C-i)\gamma + \lambda(1 - P_i(i,j+1))} + \frac{\lambda \sum_{m=i+1}^{i'} P_{i'}(m,j) P_m(i,j+1)}{\min(i,j)\mu + (C-i)\gamma + \lambda(1 - P_i(i,j+1))}$$

$$T_{i,j} = \frac{1 + (C-i)\gamma T_{i+1,j} + \lambda T_{i,j+1}}{\min(i,j)\mu + (C-i)\gamma + \lambda(1 - P_i(i,j+1))} + \frac{\lambda \sum_{m=i+1}^{C} T_{m,j} P_m(i,j+1)}{\min(i,j)\mu + (C-i)\gamma + \lambda(1 - P_i(i,j+1))}$$

$$H_{i,j} = \frac{j + (C-i)\gamma H_{i+1,j} + \lambda H_{i,j+1}}{\min(i,j)\mu + (C-i)\gamma + \lambda(1 - P_i(i,j+1))} + \frac{\lambda \sum_{m=i+1}^{C} H_{m,j} P_m(i,j+1)}{\min(i,j)\mu + (C-i)\gamma + \lambda(1 - P_i(i,j+1))}$$

$$E_{i,j} = \frac{(C-i) r_{setup} + (C-i)\gamma E_{i+1,j} + \lambda E_{i,j+1}}{\min(i,j)\mu + (C-i)\gamma + \lambda(1 - P_i(i,j+1))} + \frac{\lambda \sum_{m=i+1}^{C} E_{m,j} P_m(i,j+1)}{\min(i,j)\mu + (C-i)\gamma + \lambda(1 - P_i(i,j+1))}.$$

B. Staggered Threshold

The second policy analysed in this paper is the staggered threshold policy. While it shares some qualities with the bulk setup policy, it aims to have a more reasonable setup behaviour. That is, dynamic servers are gradually turned on as more jobs accumulate in the queue. As before, the number of static servers (C^*) is left as a decision variable. However, when there are nk jobs in the queue (waiting to be served), at least n of the dynamic servers will be busy or in setup. Formally, the number of servers in setup while in state (i,j), where $i = C^* + i'$, equals $f(C^* + i', j) = \{\lfloor \{j - C^*\}^+ / k \rfloor - i'\}^+$. Moreover, as before, a dynamic server will be switched off the moment it idles. It is worth noting that other than the lack of turning servers on in bulk, this policy does not violate any structural properties presented in [28].

To evaluate (1) and (2), all boundary condition equalities given in the bulk setup case also apply here. Furthermore, the recursive expressions for the non-repeating states, i.e. when $j < (i - C^* + 1)k + C^*$, $C^* \leq i \leq C$, and $i \leq i'$ are,

$$P_i(i,j) = \frac{\min(i,j)\mu}{\min(i,j)\mu + f(i,j)\gamma + \lambda(1 - P_i(i,j+1))}$$

$$P_{i'}(i,j) = \frac{f(i,j)\gamma P_{i'}(i+1,j)}{\min(i,j)\mu + f(i,j)\gamma + \lambda(1 - P_i(i,j+1))} + \frac{\lambda \sum_{m=i+1}^{C} P_{i'}(m,j)P_m(i,j+1)}{\min(i,j)\mu + f(i,j)\gamma + \lambda(1 - P_i(i,j+1))}$$

$$T_{i,j} = \frac{1 + f(i,j)\gamma T_{i+1,j} + \lambda T_{i,j+1}}{\min(i,j)\mu + f(i,j)\gamma + \lambda(1 - P_i(i,j+1))} + \frac{\lambda \sum_{m=i+1}^{C} T_{m,j}P_m(i,j+1)}{\min(i,j)\mu + f(i,j)\gamma + \lambda(1 - P_i(i,j+1))}$$

$$H_{i,j} = \frac{i + f(i,j)\gamma H_{i+1,j} + \lambda H_{i,j+1}}{\min(i,j)\mu + f(i,j)\gamma + \lambda(1 - P_i(i,j+1))} + \frac{\lambda \sum_{m=i+1}^{C} H_{m,j}P_m(i,j+1)}{\min(i,j)\mu + f(i,j)\gamma + \lambda(1 - P_i(i,j+1))}$$

$$E_{i,j} = \frac{\max(0, i-j)r_{idle} + f(i,j)r_{setup} + f(i,j)\gamma E_{i+1,j}}{\min(i,j)\mu + f(i,j)\gamma + \lambda(1 - P_i(i,j+1))} + \frac{\lambda(E_{i,j+1} + \sum_{m=i+1}^{C} E_{m,j}P_m(i,j+1))}{\min(i,j)\mu + f(i,j)\gamma + \lambda(1 - P_i(i,j+1))}.$$

The order in which the recursion is solved is the same as that described in the bulk setup section. That is, start with the lower right hand corner of the non-repeating portion of the CTMC, i.e. state $(C, (C - C^* + 1)k + C^* - 1)$, then proceed left along that row solving the values for each state, then move down one row to the right most state, i.e. $(C-1, (C-C^*+1)k+C^*-1)$, and repeat.

IV. Numerical Results and Observations

With the analysis complete, we proceed with our numerical experiments. Using the results from the previous section, we compute exact values for $\mathbb{E}[R]$ and $\mathbb{E}[E]$. In particular, there is no need to use simulations or approximations. All experiments were run using standard Matlab libraries, of which the source code can be found at [30]. Furthermore, each experiment evaluates the system for every valid value of C^* ($0 \leq C^* \leq C$), while each curve represents a different choice of the threshold value k. For all configurations we fix $\mu = 1$ and $\lambda = C/2$. Fixing λ does not limit the overall system behaviours, since we are interested in how the system will provision itself under a given policy and configuration (determining the expected cost metrics), and such provisioning can dynamically change the short term system load. Moreover, in [29] we found the relative load on the system to be more descriptive than the total number of servers available, therefore we also fix $C = 100$. An extensive suite of experiments can be found in [29] where some of these conditions are relaxed.

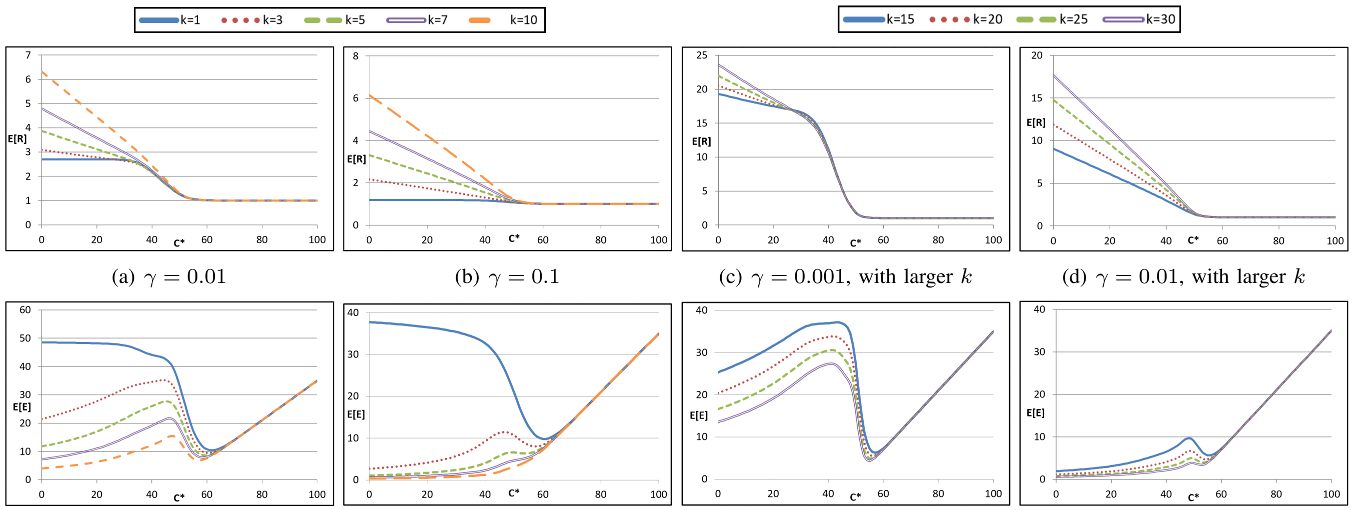
Before we proceed with our results and discussion, it is worth commenting on cost functions which are usually associated with these models. In the literature, different authors use different cost functions. As an example, the authors of [14] focus on the energy response product, i.e. $\mathbb{E}[E]\mathbb{E}[R]$, while others [9], [11] use a linear sum of the metrics, i.e. $\mathbb{E}[R] + \beta\mathbb{E}[E]$ for some $\beta > 0$. Moreover, one can define an infinite set of legitimate cost functions dependent on these metrics [23]. The problem lies in the fact that a policy or configuration which minimizes one cost function could potentially be disastrous for another. Furthermore, cost function parameters, such as the aforementioned β can often be tweaked to produce overall desired effects. One assumption which we feel justified in making however, is that all reasonable cost functions are non-decreasing in the costs. Therefore, instead of applying our numerical results to a specific cost function, we instead evaluate $\mathbb{E}[R]$ and $\mathbb{E}[E]$ separately and identify configurations which are close to simultaneously minimizing both metrics. If such win-win scenarios exist, they would minimize a large set of, if not all, well-formed cost functions.

A. Bulk Setup

We firstly inspect the behaviour of $\mathbb{E}[R]$ under the bulk setup policy. This behaviour can be seen in Figures 3 (a)-(d). As expected, $\mathbb{E}[R]$ is monotonically decreasing in C^*. However, $\mathbb{E}[R]$ has a more interesting relationship with regards to the choice of k. One would perhaps expect that the lower the value of k, the lower the expected response time would be. This is a reasonable thought since a lower value of k means a more proactive system, where servers are more inclined to turn on if there are jobs waiting. However, this is not always the case. Figures 3(a) and (c) are examples of this. Here for some lower values of C^* the expected response time for $k = 1$ is actually the largest among all curves shown. While at first perplexing, there is an intuitive explanation. While it is true that for a larger value of k the first few jobs to arrive will wait in the queue and have longer response times, this is overcome by the fact that when the server turns on, there are now more jobs to process. Because there are more jobs to process, it will take longer for the server to become idle. Due to there being a larger window for a job to arrive when the server is already on, a larger value of k can actually result in a lower expected response time.

Observation 1. *There exist system configurations where increasing the value of k decreases $\mathbb{E}[R]$.*

Looking at some curves with larger values of k, i.e. Figures 3 (d) and (h), shows another interesting behaviour. It seems that when k is sufficiently large, the expected response time decreases linearly with C^* until a point where it practically equals $1/\mu$. The point at which this changes in relation to C^* happens around $C/2$. The reason for $\mathbb{E}[R]$ converging to $1/\mu$ is clear. As the number of servers which are always on increases, the probability that the job has to wait in queue decreases, and its response time becomes its service time. On the other hand, if C^* is lower, the probability of a job



(a) $\gamma = 0.01$ (b) $\gamma = 0.1$ (c) $\gamma = 0.001$, with larger k (d) $\gamma = 0.01$, with larger k

(e) $\gamma = 0.01$ (f) $\gamma = 0.1$ (g) $\gamma = 0.001$, with larger k (h) $\gamma = 0.01$, with larger k

Fig. 3: Bulk setup $\mathbb{E}[R]$ vs C^* for (a)-(d) and corresponding $\mathbb{E}[E]$ vs C^* for (e)-(h), $C = 100$, $\lambda = 50$, $\mu = 1$

having to wait in the queue increases. While it is not entirely clear why this increase in expected response time is linear, the following is noted. When a job arrives to the system and has to wait in queue, it can be served in one of two ways. Firstly, a fresh server can turn on and begin to process the job. Secondly, a server which is currently processing a job can complete and begin to process the job which is waiting. The expected amount of time to turn on a new server increases with C^*. However, the expected time for a server to become available decreases with C^*. These two conflicting effects may counteract each other to produce a linear decrease in the expected response time of the system, in relation to C^*.

Observation 2. *For a large enough k, $\mathbb{E}[R]$ and $\mathbb{E}[E]$ can be approximated by a piecewise linear function. However, the value of k required to invoke this behaviour in the $\mathbb{E}[R]$ curve is less than the corresponding value of k for the $\mathbb{E}[E]$ curve.*

We shift our discussion to focus on the expected energy cost shown in Figures 3 (e)-(h). As a reminder, $\mathbb{E}[E]$ is the expected excess energy consumed by the system, due to servers idling and in setup. These figures show the sum of those two separate effects. Unlike the expected response time, the expected energy rate is not monotonically decreasing (nor increasing) in C^*. This leads to local maxima and minima within the curve. Firstly, it is noted that around $\rho = \lambda/\mu = C/2$ there is a local maximum. Our conjectured reason for this is the system is in a lose-lose scenario. That is, the system has a relatively high chance of being in a state where there are servers in setup, which once turned on, will clear jobs out of the system causing the relativley large set of static servers to regularily idle. Therefore, a significant amount of both setup and idling costs are contributing to the overall expected energy cost. This is in contrast to the curve around $\rho + \sqrt{\rho}$, where there is a local minimum, or in the case where γ or k is small, a global minimum. Here the system finds itself in a win-win configuration. That is, the chance of a job arriving to the system where there are no idle servers is low (consistent with the square root staffing rule [31]), and therefore the chance of servers being in setup is also low. On the other hand, the static servers are highly utilized, keeping the idling costs low. These two observations together make $C^* = \rho + \sqrt{\rho}$ an appealing choice, especially for systems with longer expected setup times.

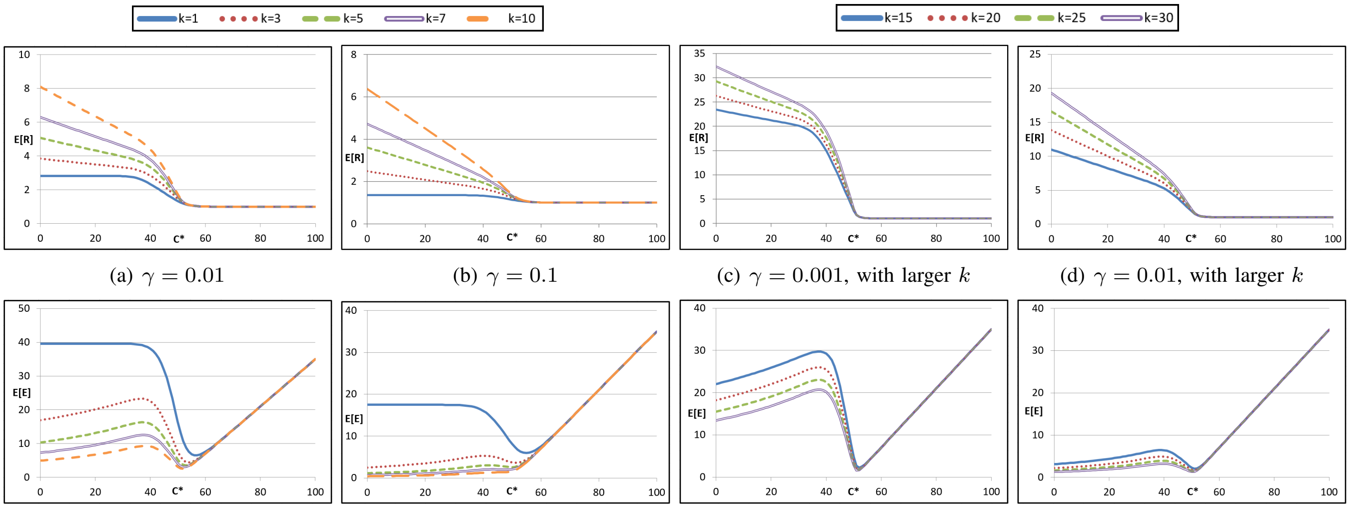
Observation 3. *For lower values of γ (longer setup times), $\mathbb{E}[E]$ has a local maximum around $C^* = \rho$.*

Looking back at the expected response time, the observation of $C^* = \rho + \sqrt{\rho}$ being a good choice for C^* also holds from the performance standpoint. The previous point that a job will rarely wait implies that the expected response time is close to its lower bound of $1/\mu$. This can be seen in Figures 3 (a)-(d). Furthermore, while the expected energy rate is sensitive to some configurations around $C^* = \rho + \sqrt{\rho}$, it is less sensitive when C^* is overestimated. Or in other words, around $C^* = \rho + \sqrt{\rho}$, $\mathbb{E}[E]$ increases at a lower rate when C^* increases, than if C^* were to decrease. This is also good news for system efficacy, as $\mathbb{E}[R]$ is monotonically decreasing in C^*. Therefore, if one wished to err on the side of caution one could set their choice of C^* to be greater than the minimum value without being punished too harshly.

Observation 4. *For low values of γ (longer setup times), the value of C^* which minimizes $\mathbb{E}[E]$, and the value of C^* which minimizes $\mathbb{E}[R]$, are approximately equal.*

B. Staggered threshold

We complete our numerical results with the *staggered threshold* policy. As discussed previously, this policy aims to capture the predictability of the bulk setup policy, while having a more appealing implementation. The first thing of note is that in general these graphs look similar to those seen



(a) $\gamma = 0.01$ (b) $\gamma = 0.1$ (c) $\gamma = 0.001$, with larger k (d) $\gamma = 0.01$, with larger k

(e) $\gamma = 0.01$ (f) $\gamma = 0.1$ (g) $\gamma = 0.001$, with larger k (h) $\gamma = 0.01$, with larger k

Fig. 4: Staggered threshold $\mathbb{E}[R]$ vs C^* for (a)-(d) and corresponding $\mathbb{E}[E]$ vs C^* for (e)-(h), $C = 100$, $\lambda = 50$, $\mu = 1$

in Section IV-A. While it is true that both policies turn servers off when they idle, it should be obvious that the staggered nature of turning servers on makes the system slower to adapt to waiting jobs or bursts of traffic. However, the majority of the observations made for the bulk setup policy hold here as well. One notable difference between these policies is that the response time does not decrease as close to linearly here as it did in the bulk setup results. Figure 4 (d) is a good example of this, in contrast to Figure 3 (d).

Observation 5. *The overall shape of the $\mathbb{E}[R]$ and $\mathbb{E}[E]$ curves is relatively insensitive to the decision of employing the bulk setup or staggered threshold policy.*

Arguably the most important similarity to that of the bulk setup policy is the presence of the aforementioned "sweet spot" in the energy curves. That is, the expected rate of energy consumption often has a minimum relatively close to $\rho + \sqrt{\rho}$, where $\rho = \lambda/\mu$, for many of the energy curves. It should be noted that for some of the energy curves for larger values of k, such as Figure 4 (f), while the minimum is actually at $C^* = 0$, the value at $\rho + \sqrt{\rho}$ is still only a slight increase from the minimum value. Therefore, for all the experiments we ran, it holds that $\rho + \sqrt{\rho}$ is a reasonable choice for C^* with regards to energy costs as well as system performance, for reasons argued previously. Moreover, inspecting the choice of k for this value of C^* leads to an interesting implication.

Observation 6. *The expected energy costs for the bulk setup and staggered threshold policies are decreasing in k.*

Reviewing Figures 4 (e)-(h) one will note that for all fixed values of C^* the expected energy cost is decreasing in k. That is, the longer the system is willing to wait before turning servers on, the lower the energy costs will be. This is an intuitive result, but perhaps not obvious. Consider the following fallacious argument. If k is large, the system could be put in a situation where there are a lot of excess jobs in the system by the time the next server completes its setup, this will cause a greater number of servers to be turned on in the short run. Due to this large number of servers now on, the system will quickly clear out all of the current jobs. Jobs departing from the system due to dynamic servers being turned on will now cause static servers to become idle where they otherwise may have been busy, thus incurring a higher expected energy cost. However, from our numerical results we can see that this is not the case (at least for the parameters we examined). The reason the energy costs are lower for higher values of k is that dynamic servers are less likely to "thrash". For example, if a server begins its setup when there is one job waiting ($k = 1$), it will incur an initial setup cost in the short run that it may otherwise not for a larger value of k, but it may also quickly clear the job out, switch off, and then find itself in the same situation of one job waiting to be served in the near future. This causes multiple setup cycles to occur to deal with a set of jobs which a higher value of k may deal with using only a single setup, or potentially without any setups at all. Due to a lower number of server setups for a higher value of k, the expected energy cost is strictly lower. Therefore, if energy costs are the only concern, one should choose the highest possible value of k. One needs to be careful however, since higher values of k could have a (potentially disastrous) negative impact on performance. After further though, this may not be the case pertaining to the choice of $C^* = \rho + \sqrt{\rho}$. Viewing Figures 3 and Figures 4 (a)-(d), one notes that around $C^* = \rho + \sqrt{\rho}$ the expected response time is quite insensitive to the choice of k. Therefore, the largest possible value of k should be chosen. Since there is no restriction on the ceiling of k, one should let $k \to \infty$. If that is the case however, the system degenerates to the well known $M/M/C^*$ queueing system where $C^* = \rho + \sqrt{\rho}$.

Observation 7. *For all parameter configurations examined here, for both the expected response time and expected energy costs, the degenerate solution of using an $M/M/C^*$ queue is near-optimal for some C^* around $\rho + \sqrt{\rho}$.*

While perhaps at first this is a disappointing result, since it implies energy costs cannot be saved, it gives an elegant and simple solution to what on the surface, appears to be a complex problem. We argue that for linear cost functions the bulk setup policy is a reasonable approximation of the optimal policy, see [28]. However, the bulk setup turn on criteria hinges on interruptible setups and exponentially distributed setup times. We therefore in turn analyse the staggered threshold policy. We find that an $M/M/C^*$ queue is close to optimal for both of these policies. Thus, we argue that an $M/M/C^*$ queue is close to optimal across all potential policies for some C^*. Furthermore, this observation is consistent with the contributions of [14] where they present a similar square root provisioning result for the particular case of the *staggered setup* policy (staggered threshold with $k = 1$ and $C^* = 0$) under the cost function $\mathbb{E}[E]\mathbb{E}[R]$.

These results would suggest that near optimal control of these multiserver systems can be achieved with a single decision variable, C^*. Moreover, the choice of C^* is solely dependent on ρ. In other words, to have a near optimal system, one need only concern themselves with accurately determining λ and μ (and not potentially complicated and convoluted setup and turn off criteria). Such a solution offers another benefit as well. Researchers often choose to incorporate the expected *rate of switching* (how often servers turn on/off) to capture the wear and tear cost of the hardware [11], [23], [32]. It immediately follows that this cost metric is trivially minimized when only a static allocation of servers is employed. Therefore, any well-formed cost function including the expected rate of switching also agrees with the degenerate solution.

The argument of an $M/M/C^*$ queue being a near optimal solution is further enforced by revisiting Observation 6 in more detail. Observation 6 tells us that to minimize the expected energy cost, the best choice of k is the largest value of k, or $k = 30$ if limited to the choice of our experimental parameters. But if the system is stable, specifically if the system has approximately $\rho + \sqrt{\rho}$ static servers, what is the physical interpretation of such a large value for k? Clearly, the probability that there are greater than n jobs in the system for our model, is less than or equal to the probability that there are greater than n jobs in a classic $M/M/C^*$ queue. That is, $P(N > n) < P(N_{M/M/C^*} > n)$, where $N_{M/M/C^*}$ is a random variable denoting the number of jobs in an $M/M/C^*$ queue, and $C^* < C$. But using $C^* = \lceil \rho + \sqrt{\rho} \rceil = 58$ and $C = 100$, one can do a quick calculation to find that $P(N > 87) < 0.0023$. In other words, if $k = 30$, at least 434 jobs out of 435 will not cause the first dynamic server to begin its setup process when they arrive. Furthermore, approximately only 1 job out of every 44,000 has a chance of initiating the setup process of the second dynamic server when it arrives. Therefore, the physical interpretation that larger values are a good choice for k corresponds to saying the system should not utilize its dynamic servers, but instead be statically provisioned. Again, this gives rise to a simple and easy to implement solution.

V. Conclusion

Provisioning server farms and datacenters is an actively studied and open problem in the intersection of green computing and queueing theory. We presented a well-established model which views these server farms as a multiserver queueing system with setup times. From this model we studied two specific policies, *bulk setup*, and *staggered threshold*. Using the recursive renewal reward technique, we performed an exact analysis for each of these policies. That is, we were able to arrive at exact expressions for the expected response time and expected energy costs for the two aforementioned policies. Using these expressions, we performed an extensive numerical analysis examining how these metrics behave with respect to system parameters, and underlying decision variables. From this numerical analysis we discovered and commented on several interesting observations which grant insight into how these systems behave. This includes, but is not limited to, our argued degenerative solution that an $M/M/C^*$ queue is reasonably close to optimal across all potential policies for some choice of C^* around $\rho + \sqrt{\rho}$.

Moving forward with our research we wish to formally show the asymptotic equivalence of the bulk setup, staggered threshold, and all other threshold policies which allow for a static number of servers to be provisioned. This would give an analytical result which would bridge the gap from the known optimal bulk setup policy to the observed near optimal results of staggered threshold policy (among others). Furthermore, we would like to inspect the sensitivity of these results to estimating a time varying arrival rate ($\lambda(t)$) where static servers are provisioned on a macro scale dependent on the setup rate γ.

Acknowledgement

This research was funded by the Natural Sciences and Engineering Research Council of Canada.

References

[1] EPA, "Report to congress on server and data center energy efficiency," tech. rep., U.S Environmental Protection Agency, 2007.

[2] J. Koomey, "Growth in data center electricity use 2005 to 2010." A report by Analytical Press, completed at the request of The New York Times, http://www.analyticspress.com/datacenters.html, 2011.

[3] D. Paul, W. D. Zhong, and S. K. Bose, "Energy efficient scheduling in data centers," in *International Conference on Communications*, IEEE, pp. 5948–5953, 2015.

[4] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," vol. 39, no. 4, pp. 123–134, 2009.

[5] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[6] Y. Peng, D. K. Kang, F. Al-Hazemi, and C. H. Youn, "Energy and QoS aware resource allocation for heterogeneous sustainable cloud datacenters." *Optical Switching and Networking*, Preprint, 2016.

[7] B. Yang, Z. Li, S. Chen, T. Wang, and K. Li, "Stackelberg game approach for energy-aware resource allocation in data centers." *Operations Research Letters*, Preprint, 2016.

[8] M. Callau-Zori, L. Arantes, J. Sopena, and P. Sens, "Merci-miss: Should I turn off my servers?," in *Distributed Applications and Interoperable Systems*, pp. 16–29, 2015.

[9] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. H. Andrew, "Greening geographical load balancing," in *the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pp. 233–244, 2011.

[10] M. Mazzucco and D. Dyachuk, "Optimizing cloud providers revenues via energy efficient server allocation," *Sustainable Computing: Informatics and Systems*, vol. 2, no. 1, pp. 1–12, 2012.

[11] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," *SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, pp. 303–314, 2005.

[12] J. Slegers, N. Thomas, and I. Mitrani, "Dynamic server allocation for power and performance," in *SPEC International Workshop on Performance Evaluation: Metrics, Models and Benchmarks*, pp. 247–261, 2008.

[13] N. Tian and Z. G. Zhang, *Vacation Queueing Models - Theory and Applications*. Springer Science, 2006.

[14] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Performance Evaluation*, vol. 67, no. 11, pp. 1155–1171, 2010.

[15] A. Gandhi, M. Harchol-Balter, and I. Adan, "Server farms with setup costs," *Performance Evaluation*, vol. 67, no. 11, pp. 1123–1138, 2010.

[16] A. Gandhi, S. Doroudi, M. Harchol-Balter, and A. Scheller-Wolf, "Exact analysis of the M/M/k/setup class of Markov chains via recursive renewal reward," in *ACM SIGMETRICS Performance Evaluation Review*, pp. 153–166, 2013.

[17] T. Phung-Duc, "Exact solutions for M/M/c/setup queues," *arXiv:1406.3084*, 2014.

[18] S. Doroudi, B. Fralix, and M. Harchol-Balter, "Clearing analysis on phases: Exact limiting probabilities for skip-free, unidirectional, quasi-birth-death processes," *arXiv preprint arXiv:1503.05899*, 2015.

[19] X. Xu and N. Tian, "The M/M/c queue with (e, d) setup time," *Journal of Systems Science and Complexity*, vol. 21, no. 3, pp. 446–455, 2008.

[20] P. J. Kuehn and M. E. Mashaly, "Automatic energy efficiency management of data center resources by load-dependent server activation and sleep modes," *Ad Hoc Networks*, vol. 25, no. 2, pp. 497–504, 2015.

[21] Y. Ren, T. Phung-Duc, Z. W. Yu, and J. C. Chen, "Design and analysis dynamic auto scaling algorithm (DASA) for 5G mobile networks," *arXiv preprint arXiv:1604.05803*, 2016.

[22] J. R. Artalejo, "A unified cost function for M/G/1 queueing systems with removable server.," *Trabajos de Investigacion Operativa*, vol. 7, no. 1, pp. 95–104, 1992.

[23] V. J. Maccio and D. G. Down, "On optimal policies for energy-aware servers," *Performance Evaluation*, vol. 90, pp. 36 – 52, 2015.

[24] M. E. Gebrehiwot, S. Aalto, and P. Lassila, "Optimal sleep-state control of energy-aware M/G/1 queues," in *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, pp. 82–89, 2014.

[25] M. E. Gebrehiwot, S. Aalto, and P. Lassila, "Energy–performance trade-off for processor sharing queues with setup delay," *Operations Research Letters*, vol. 44, no. 1, pp. 101–106, 2016.

[26] E. Hyytiä, R. Righter, and S. Aalto, "Task assignment in a heterogeneous server farm with switching delays and general energy-aware cost structure," *Performance Evaluation*, vol. 75–76, pp. 17–35, 2014.

[27] E. Hyytiä, R. Righter, and S. Aalto, "Energy-aware job assignment in server farms with setup delays under LCFS and PS," in *26th International Teletraffic Congress (ITC 26)*, pp. 1–9, 2014.

[28] V. J. Maccio and D. G. Down, "On optimal control for energy-aware queueing systems," in *27th International Teletraffic Congress (ITC 27)*, pp. 98–106, 2015.

[29] V. J. Maccio and D. G. Down, "Exact analysis of energy-aware queueing policies," Tech. Rep. CAS-16-01-DD, Department of Computing and Software, McMaster University.

[30] "Source code." http://www.cas.mcmaster.ca/~macciov/publications.html. Accessed: 2016-03-01.

[31] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[32] T. H. Nguyen, M. Forshaw, and N. Thomas, "Operating policies for energy efficient dynamic server allocation," *Electronic Notes in Theoretical Computer Science*, vol. 318, pp. 159–177, 2015.