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

V. J. Maccio[a], D. G. Down[a]

[a]*McMaster University, Hamilton, Ontario*

Abstract

Energy consumption of today's datacenters is a constant concern from the standpoints of monetary and environmental costs. An intuitive solution to address these immense energy demands is to turn servers off to incur less costs. As such, many different authors have modeled this problem as an $M/M/C$ queue where each server can be turned on, with an exponentially distributed setup time, or turned off instantaneously. What policy the model should employ, or rather when each server should be turned on and off is far from a trivial question. A specific policy is often examined, but determining which policy to study can be a difficult process and is often a product of intuition. Moreover, while a specific policy may do comparatively well against another, in general it may be far from optimal. This problem is further accentuated when one considers the case that a policy may do well or even be optimal under a specific cost function, but far from optimal under another. To address this issue we study the structural properties of the optimal policy under linear cost functions, allowing for a significant reduction in the search space. We then leverage these structural properties to intelligently select two policies for further study. 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.

Keywords: Continuous time Markov chain, Markov decision process, green computing, energy-aware server, queueing analysis

1. 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 [8, 20]. Google alone pays an annual energy bill on the order of tens of millions of dollars [35]. 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 [4]. 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.

Such concerns have driven an interest in queueing systems where individual servers can be turned on to improve performance, and turned off to save on costs. This interest has led to different authors studying the same, or similar, queueing models. However, due to the complexity of the problem, i.e. the choice of cost function, policy implemented, model details, etc., different conclusions can be drawn from similar underlying problems. One consequence of the variety in the problems studied and the corresponding variety of insights is that it is difficult to confidently draw conclusions which are overarching across the problem domain. To address this issue, this work presents a two-pronged approach. Firstly, we derive several structural properties of the optimal policy. These properties allow one to discount policies which are in turn known to exhibit sub-optimal behaviours, as well as gain confidence in previously studied policies which adhere to this structure. Secondly, we leverage these structural properties to intelligently select two policies to analyse further. We

perform an exact analysis of these policies which grants insights into how these systems behave and how they should be provisioned. Specifically, we provide insight on how many servers should always remain on under any reasonable cost function.

To the best of our knowledge, Chen et al. [6] 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. [37] 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 [39] 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 [10] and provided several analytical results for the single server case, as well as some rules of thumb for the multiserver case. They continued their research in [11, 12] in which they modelled a server farm as a continuous time Markov chain (CTMC) employing the *staggered setup* policy, where the number of servers in setup equals the number of jobs waiting to be served and servers 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 [9] 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 applied their method of analysis to the *delayed off* policy, an extension of staggered setup where servers spend some exponentially distributed period of time idle before being switched off. Phung-Duc [33] gave a comprehensive side by side comparison of RRR and other traditional methods for analysing these CTMCs. If the steady state distribution of these CTMCs is also of interest, methods introduced by Doroudi et al. in [7] may be employed.

Other authors have studied the same model as Gandhi et al. but under different policies (when servers turn on and off). Mitrani [30] studied this model where a reserved set of servers are brought into setup when the number of jobs in the system exceeds a threshold, and then shuts those servers off once the number of jobs drops below another threshold. This policy was further studied in [17]. Xu and Tian [40] studied the set of policies where e servers are turned off when there are d servers idle. Kuehn and Mashaly [21] analysed policies which wait for a threshold number 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. [36] 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 [3] 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 [25] 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. [14] 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 [15]. Hyytiä et al. [18, 19] also studied this model under processor sharing in addition to *last come first serve*, and different routing configurations. This model also has applications to or is studied in problems which arise in other fields such as manufacturing, logistics, and vacation models [2, 28, 38]. For other, non-queueing theoretic approaches to this problem see [5, 22, 29, 32, 41].

While the contributions of the previously mentioned works are substantial, a gap in knowledge still remains. While optimal control in the single server case is well understood, it offers less practical application than corresponding multiserver models. However, when studying the multiserver case, complexity has constrained researchers to focus on specific policies, which in general may be far from optimal. Moreover, when evaluating one of these policies it may do well for a specific cost function, but poorly under another. Therefore, saying one policy is strictly better than another, or any general statements, can be difficult. Reiterating, to address these issues this work includes but is not limited to the following contributions:

1. An examination of the optimal policy and behaviour under linear cost functions, including formal proofs of several key structural properties.

2. The description and exact analysis of two distinct policies which leverage the aforementioned struc-

tural properties, *bulk setup* and *staggered threshold*.

3. A range of numerical experiments which yield exact values for metrics of interest alongside 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 conference publications and technical reports [23, 24, 27].

2. Model

The model under study is an $M/M/C$ queue where each server can be switched on and off, and where turn-offs are instantaneous, but turn-ons take an exponentially distributed setup time. This is described formally as follows. Jobs arrive to a central queue following a Poisson process with rate λ, are processed on a first come first served basis, and have processing times (job sizes) which are exponentially distributed with rate μ. Furthermore, there are C homogeneous servers present, each of which can be in one of four energy states: *off, setup, idle*, or *busy*. For ease of exposition this work often refers 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, when a server is *off* it may begin turning on by moving to *setup*. Once in *setup* the server will remain there for an exponentially distributed amount of time with rate γ before it is turned on and becomes *idle* or *busy*. When on, the server is *idle* if it is not processing a job and *busy* if it is. A server is free to move between *idle* and *busy* as long as there is a job present which it could serve. Furthermore, at any time a server can instantly be switched *off*. It is worth noting that this implies a server's setup process can be canceled when a server is turned *off*. The model of study can be viewed in Figure 1.

Formally an energy-aware system can be viewed as a four-tuple $(C, \lambda, \mu, \gamma)$, where C is the number of servers, and λ, μ, and γ are the arrival, processing, and setup rates, respectively. The system load ρ is defined as $\rho = \lambda/(C\mu)$, and is assumed to be strictly less than one. These four underlying parameters of an energy-aware system are usually thought to be given, that is to say there does not exist any control to mutate these values. One does have control however, of when each of these C servers are turned on and off. A precise description of this server behaviour is referred to as a policy. In this work a specific policy is denoted by π. A policy can be given explicitly. For example, for a two server system ($C = 2$), one could describe a policy as: turn the first server on if there are two or more jobs in the system, turn the second server on if there are five or more jobs in the system, turn servers off when the system becomes empty. Of course as C gets larger describing a policy in this way can lead to an unmanageable number of parameters. As such, policies are often described in a broader fashion. Some examples of these descriptions are: the number of servers which are on or in setup equals the number of jobs in the system, turn on all servers once there are k jobs in the system and turn them off when they idle, keep all servers on all the time, etc. Often times when describing these policies a set of servers is provisioned which always remain on. We refer to a server which always remains on as *static*. Furthermore, a server which can be turned off and on is referred to as *dynamic*.

The set of potential policies one can study regarding these energy-aware systems is infinite, and the natural desire is to compare policies against each other. But in order for this to be possible, we need to associate metrics with an energy-aware system employing a given policy. In other words, one must define what it means to be better. This work examines the trade-off between efficacy and efficiency. The expected response time, denoted by $\mathbb{E}[R]$, is employed to evaluate efficacy, while the expected energy cost, denoted by $\mathbb{E}[E]$, is employed to evaluate efficiency. The expected response time is the expected amount of time a job spends in the system, from arrival to departure. The expected energy cost takes a little more care to define. Each of the energy states (*off, idle, busy*, and *setup*) has a corresponding energy consumption rate. Let these rates be denoted by E_{Off}, E_{Idle}, E_{Busy}, and E_{Setup}, respectively, where $E_{\text{Off}} < E_{\text{Idle}} < E_{\text{Setup}}$ and $E_{\text{Idle}} < E_{\text{Busy}}$ (note that no assumption is imposed on the ordering of E_{Busy} relative to E_{Setup}). Furthermore,

let the random variables C_{Off}, C_{Idle}, C_{Busy}, and C_{Setup} denote the number of servers which are off, idle, busy, or in setup, respectively. Then

$$\mathbb{E}[E] = E_{\text{Off}}\mathbb{E}[C_{\text{Off}}] + E_{\text{Idle}}\mathbb{E}[C_{\text{Idle}}] + E_{\text{Busy}}\mathbb{E}[C_{\text{Busy}}] + E_{\text{Setup}}\mathbb{E}[C_{\text{Setup}}], \tag{1}$$

where it is assumed throughout this work that $E_{\text{Off}} = 0$. This could be relaxed to account for lower energy consumption states where the server cannot process jobs, e.g. sleep states.

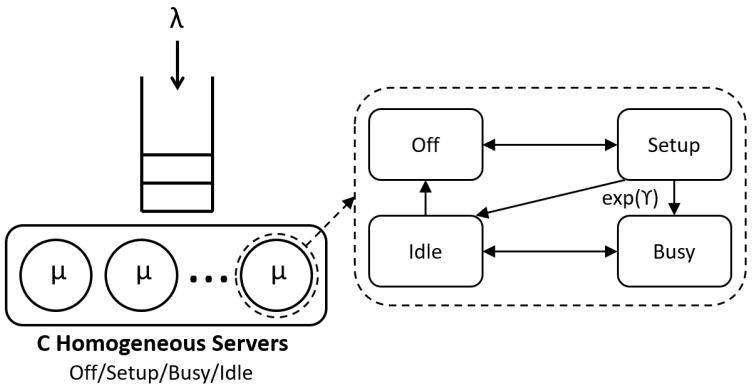


Figure 1: The model under study. Dynamic servers take time exponentially distributed with rate γ to move from *setup* to *idle* or *busy*, all other transitions happen instantly if the system state allows it.

With the cost metrics defined, one can then create a cost function dependent on these metrics and begin to compare policies. There is no limit to the number of cost functions which can be defined, but common cost functions do arise in the literature, e.g. $\mathbb{E}[R]\mathbb{E}[E]$, and $\mathbb{E}[R] + \beta\mathbb{E}[E]$. Due to the diversity of the set of possible cost functions, conclusions which span across many cost functions are often difficult to make, and moreover, niche behaviours are often easy to invoke by tweaking parameters within a cost function, such as β in $\mathbb{E}[R]+\beta\mathbb{E}[E]$. For example, one could imagine a system with a small number of servers where the policy which keeps all servers on would be optimal when β is small, since this minimizes the expected response time. But when β is large, the optimal policy for the same system may be one where servers are kept off for long periods of time while they wait for a large number of jobs to accumulate before expending the energy to turn on. As such, this work strives to draw conclusions applicable to large sets of cost functions, and in fact does so for all *well-formed cost functions*.

Definition 1. ***Well-Formed Cost Function:*** *A cost function $\mathcal{C}(\cdot)$ is a well-formed cost function if it is non-decreasing in, dependent on, and only dependent on, the expected response time, i.e. $\mathbb{E}[R]$, and the expected energy costs, i.e. $\mathbb{E}[E]$.*

If one can construct a policy such that $\mathbb{E}[R]$ and $\mathbb{E}[E]$ are reasonably close to their minimum values, then that policy should do reasonably well under any well-formed cost function. If such policies exist is far from a trivial question. The issue is made more complex after realizing that such a policy may exist for a specific energy-aware system, but could be a disaster for another. For instance one policy may always do well if setup times are exceedingly long, but could do poorly if setup times are short or even moderate in length. For analytic tractability, often times it is necessary restrict the set of well-formed cost functions to those which are linear in $\mathbb{E}[R]$ and $\mathbb{E}[E]$, i.e. cost functions of the form

$$\mathcal{C}(\mathbb{E}[R], \mathbb{E}[E]) = \mathbb{E}[R] + \beta\mathbb{E}[E],$$

where β is some scaling constant greater than 0. This subset of well-formed cost functions is referred to as the set of linear well-formed cost functions and is specifically considered when deriving structural properties of the optimal policy.

3. Structural Properties

Before one can begin to confidently construct policies which perform well for a large portion of well-formed cost functions, one must first understand what behaviours and properties a favourable policy exhibits under linear well-formed cost functions. These properties are derived by examining the optimal policy and proving several key structural properties. The proofs of all theorems presented here can be found in Appendix A.

Theorem 1. *For all energy-aware systems, for all linear well-formed cost functions, the optimal policy is a threshold policy.*

Specifically, a threshold policy is as follows. For $1 \leq m \leq C$, and $0 \leq i < m$, the mth server will have a threshold value $k^+_{m,i}$ which indicates that if there are at least $k^+_{m,i}$ jobs in the system while there are i servers currently on, and if the mth server is currently off or in setup, then it will switch to or remain in setup. Conversely, if there are less than $k^+_{m,i}$ jobs in the system and the mth server is currently off or in setup, then it will switch to or remain off. Moreover, for $1 \leq m \leq C$, the mth server will have a threshold value k^-_m which indicates if there are less than or equal to k^-_m jobs and the mth server is on, then that server will immediately switch off. Due to the two separate thresholds, in general these policies are hysteretic in nature.

The thresholds are defined such that if $m < m'$, then $k^-_m \leq k^-_{m'}$ and $k^+_{m,i} \leq k^+_{m',i}$ for any $i < m$. Furthermore, for all $i < C$, $k^-_i < k^+_{i+1,i}$. A consequence of this ordering is that the turn on thresholds do not depend on the number of servers currently in setup. Specifically, if there are $k^+_{m,i}$ jobs in the system, and the first i servers are on, then the next $(m-i-1)$ servers are in setup. For example, if $k^+_{3,1} = 4$, there is one server currently on, and there are currently four jobs in the system, then it must be true that the second server is in setup. Also, the turn off thresholds do not depend on the number of servers currently on, as this information can be inferred from the definition. That is, if the mth server is currently on, then the first $(m-1)$ servers must also be on.

Considering Theorem 1 and viewing each threshold as a decision variable, one could always consider searching for the optimal policy by running a numerical technique such as policy iteration. This approach is quickly made infeasible however, by the number of decision variables and more to the point, from the complexity of analysing even a single choice of these decision variables. That is, there are a total of $(C(C+1)+2C)/2$ decision variables, and to perform an exact analysis on one instantiation has complexity $O(C^3 k^+_{C,C-1})$. Therefore, it is important to understand the optimal policy further to construct near-optimal policies that use significantly fewer decision variables.

Theorem 2. *For all energy-aware systems, for all cost functions of the form*

$$\mathbb{E}[R] + \beta \mathbb{E}[E],$$

where $\beta \lambda E_{Idle} < 1$, if the number of jobs in the system is greater than or equal to the number of servers currently turned on, it is always suboptimal to turn a server off, which is already on.

This theorem gives an upper bound on the turn on thresholds. That is, $k^-_m < m$, whereas without Theorem 2 they can be arbitrarily large. Therefore, this theorem significantly reduces the search space for an optimal policy. While Theorem 2 gives concrete guidance as to when to turn servers off for applicable cost functions, it remains to determine criteria regarding turning servers on. As such, we introduce the notion of a *bulk setup* scheme. Here, when the system determines to turn a server on, instead of just putting one server in setup, it instead puts all available servers into setup. Recall the system can cancel setups. It then follows that once one of the servers completes its setup, the system can simultaneously cancel all remaining setups if it chooses to. One may see this as incurring unnecessary costs, but it offers advantages as well. Where before there were $C(C+1)/2$ turn on thresholds, the bulk setup scheme decreases this to C. If there are m servers currently on, where $0 \leq m < C$, then there is a single threshold k^+_m such that if there are greater than or equal to k^+_m jobs in the system then the remaining $(C-m)$ servers are in setup. Conversely, if there are less than k^+_m jobs in the system then the remaining $(C-m)$ servers are off. An arguably surprising result of this scheme is that under an appropriate cost function, there are no disadvantages.

Theorem 3. *For all energy-aware systems, for all linear well-formed cost functions the optimal policy turns servers on following a bulk setup scheme.*

It is worth noting that while bulk setup is a property of the optimal policy (assuming a linear cost function) it is highly dependent on the setup times being exponentially distributed. For example, if the setup times follow a degenerate distribution, i.e. the setup times are constant, such a scheme is often times the worst possible choice. It remains however, that one can gain insights by studying such a policy. For a further examination of these structural properties we direct the reader to [24] where several other structural properties are presented, as well as a consideration of the theorems presented here if setups cannot be canceled.

4. Exact Analysis

With structural properties of the optimal policy presented, we proceed with the exact analysis of two specific policies, *bulk setup* and *staggered threshold*. These policies are formally defined as follows.

Definition 2. ***Bulk Setup Policy:*** *A policy is a bulk setup policy if it has two decision variables, C_S and k, where C_S denotes the number of servers which always remain on, i.e. the number of static servers, and k is a threshold variable such that dynamic servers behave in the following manner:*

- *For all m and i where $C_S < m \leq C$, and $C_S \leq i < m$, $k^+_{m,i} = C_S + (i - C_S + 1)k$. In other words, if there are ever $C_S + (i - C_S + 1)k$ or more jobs in the system, where i is the number of servers currently on, then all remaining servers will be in setup, and if the number of jobs in the system is less than $C_S + (i - C_S + 1)k$, then all servers which are not already on, will be switched off (have their setups cancelled).*
- *For all m where $C_S < m \leq C$, $k^-_m = m - 1$. That is, servers turn off the moment they idle.*

It should be noted that the bulk setup policy is a threshold policy, complying with Theorem 1; it never turns a server off if there is a job it could be processing, complying with Theorem 2; and it turns servers on following a bulk setup scheme which, from Theorem 3 is known to be optimal for linear well-formed cost functions. An intuitive way to think about the policy is that each of the $(C - C_S)$ dynamic servers is responsible for k jobs once all of the static servers are busy. As an example, if there are $(C_S + 2k)$ jobs present and the second server is currently off, the system will begin turning on all available dynamic servers, and then cancels, if appropriate, the remaining setups the moment one of the servers turns on. Here only a single threshold variable is employed in contrast to the many threshold variables of a general threshold policy. We argue however, that this approximation does not lead to great loss in desirable system behaviours. If a general threshold policy had lower thresholds for the first few servers and higher thresholds for the others, a bulk setup policy could emulate this behaviour by keeping the low threshold servers always on, and choosing an appropriate single threshold to approximate the rest. Furthermore, it will be seen via our numerical experiments, an optimal bulk setup policy exhibits behaviour close to an optimal policy.

Although this policy adheres to the structural properties presented in Section 3, the bulk setup nature is admittedly unappealing from an implementation standpoint. As such the second policy we analyse holds true to Theorems 1 and 2 while turning servers on in a more appealing fashion.

Definition 3. ***Staggered Threshold Policy:*** *A policy is a staggered threshold policy if it has two decision variables, C_S and k, where C_S denotes the number of servers which always remain on, i.e the number of static servers, and k is a threshold variable such that dynamic servers behave in the following manner:*

- *For all m and i where $C_S < m \leq C$, and $C_S \leq i < m$, $k^+_{m,i} = C_S + (m - C_S)k$. This implies that if there are i servers on, and j jobs in the system, the number of servers in setup is given by $f(i,j) = \{\lfloor \{j - C_S\}^+ / k \rfloor - (i - C_S)\}^+$.*
- *For all m where $C_S < m \leq C$, $k^-_m = m - 1$. That is, servers turn off the moment they idle.*

This can again intuitively be thought of as each of the dynamic servers being responsible for k jobs when all the static servers are busy. The difference from the bulk setup policy is this policy gradually begins to turn more servers on instead of all at once. For example, if all dynamic servers are off and there are $C_S + k$ jobs in the system, exactly one server will be moved to setup, if there are $C_S + 2k$ jobs in the system then two servers will be in setup, and so on.

Bulk setup, staggered threshold and any optimal policy are all threshold policies. If an energy-aware system is implementing a threshold policy, due to the underlying exponential distributions the model can be described as a continuous time Markov chain. The state space of this CTMC is (i, j) where i denotes the number of servers currently on and j denotes the number of jobs in the system. Furthermore, which threshold policy is being implemented will determine how and when the Markov chain transitions through its rows. This results in a quasi birth death process. As a shorthand, when referring to row n (column n), we are implicitly referring to the row corresponding to $i = n$ (the column corresponding to $j = n$). A graphical representation of such a CTMC can be seen in Figure 2.

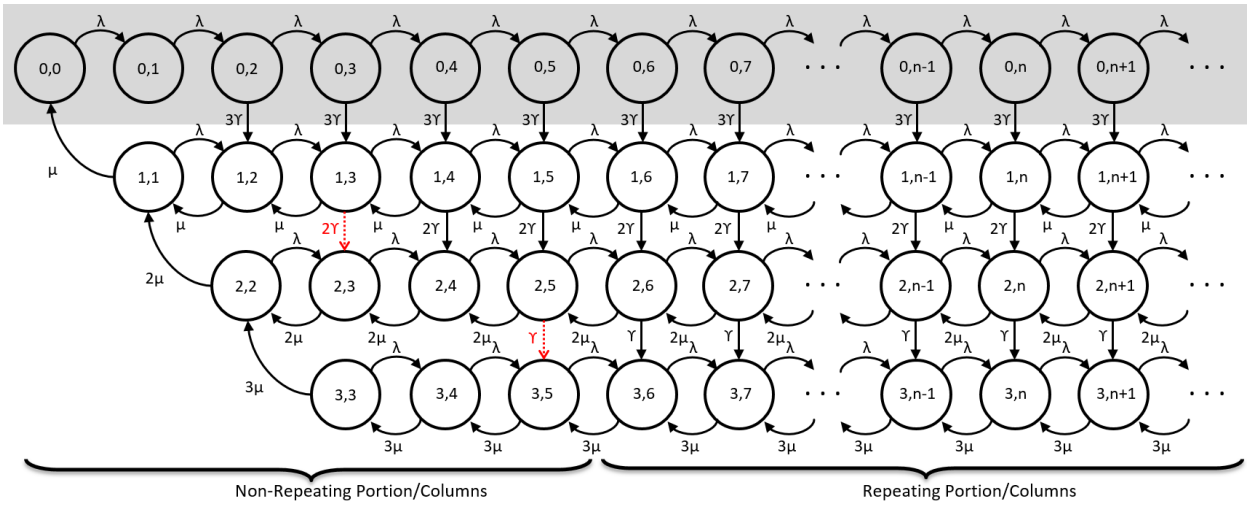


Figure 2: Bulk setup CTMC with $C_S = 0$ and $k = 2$. If C_S is changed from 0 to 1, the shaded row would be merged into row 1 creating the state $(1, 0)$ as well as adding the dashed red arrow transitions to states $(1, 3)$ and $(2, 5)$. Moreover, the repeating portion of the CTMC would now start at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

Due to the structure of these CTMCs, they can be analysed using the RRR method, formally described in [9], which allows for the exact analysis of the expected energy cost and the expected response time. The idea of the method is to build recursions for costs based on how much 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 (at rate one per job), 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 transition one column left has been made. Therefore, we denote by $P_{i'}(i, j)$ the probability of being in row i' after moving one column left of state (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 before it starts repeating. Again, using Figure 2 as a visual reference, the repeating portion starts with column 6, and continues 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 $(3, 5)$.

4.1. Derivations and Analysis

Before starting our analysis, we first make a simplification with regards to the definition of $\mathbb{E}[E]$, i.e. (1). It is noted that because each arriving job will eventually be served, the expected amount of energy spent processing jobs is independent of the policy being employed. That is, the only energy costs which are dependent on the policy are those associated with idle servers and server setups. Therefore, to highlight the differentiation between policies, for the remainder of this section energy costs contributed by busy servers are disregarded.

Assuming C_S denotes the number of static servers, we develop several expressions that are independent of which threshold policy is being employed. 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 $(C_S, 0)$, i.e. when the system is empty. Combining this with Little's law we can write:

$$\mathbb{E}[R] = \frac{\mathbb{E}[N]}{\lambda} = \frac{H_{C_S,1}}{\lambda(T_{C_S,1} + 1/\lambda)}. \tag{2}$$

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

$$\mathbb{E}[E] = \frac{E_{C_S,1} + (E_{\text{Idle}} C_S)/\lambda}{T_{C_S,1} + 1/\lambda}. \tag{3}$$

It is also noted that the underlying CTMCs of all threshold policies have identical repeating portions. Therefore, we can derive all values associated with the repeating portion independent from the choice of policy. These values are as follows,

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

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

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

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

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

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 $H_{i,j+1} = H_{i,j} + T_{i,j}$.

4.2. Bulk Setup - Analysis

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. 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_S.

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 [23] where all derivations are given in full. Firstly consider the case where $j < (i - C_S + 1)k + C_S$ or $i = C$, 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 the 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 it can be seen that when arriving to 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. 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 an 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 is easily expressed in terms of 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 $j < (i - C_S + 1)k + C_S$ or $i = C$:

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

All that is required to derive an expression for the case where $j \geq (i - C_S + 1)k + C_S$ and $i < C$ 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,

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

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 glance, 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 $T_{2,6}$, $T_{3,6}$, 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 can be drastically reduced from solving each value simultaneously via a system of linear equations. This is done by 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 $(C, (C - C_S + 1)k + C_S - 1)$ in the general case (assuming $C_S < C$), 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 on the order of $(C - C_S)^2 k$ states in the non-repeating portion, such a recursion solves all $T_{i,j}$ values with complexity $O((C - C_S)^3 k)$, as opposed to the complexity of simultaneously solving the system of equations, which is $O(((C - C_S)^2 k)^3)$.

To summarize, we give all recursions and information needed to evaluate equations (2) and (3). Firstly, due to the servers turning off when idle, the following boundary conditions are known: $(\forall i > C_S : P_{i-1}(i, i) = P_{i-1}(i - 1, i) = 1)$ and $(\forall j \leq C_S : P_{C_S}(C_S, j) = 1)$. Secondly, we present all expressions pertaining to the non-repeating portion of the chain where no servers are in setup, i.e. when $j < (i - C_S + 1)k + C_S$, $C_S \leq i \leq C$, and $i \leq i'$. They are as follows:

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

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

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

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

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

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

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

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

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

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

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

4.3. Staggered Threshold - Analysis

To evaluate the expressions for the expected response time and energy cost, i.e. (2) and (3), all boundary conditions given in the bulk setup case also apply here. Furthermore, the recursive expressions for the non-repeating states, i.e. when $j < (i - C_S + 1)k + C_S$, $C_S \leq i \leq C$, and $i \leq i'$ are,

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

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

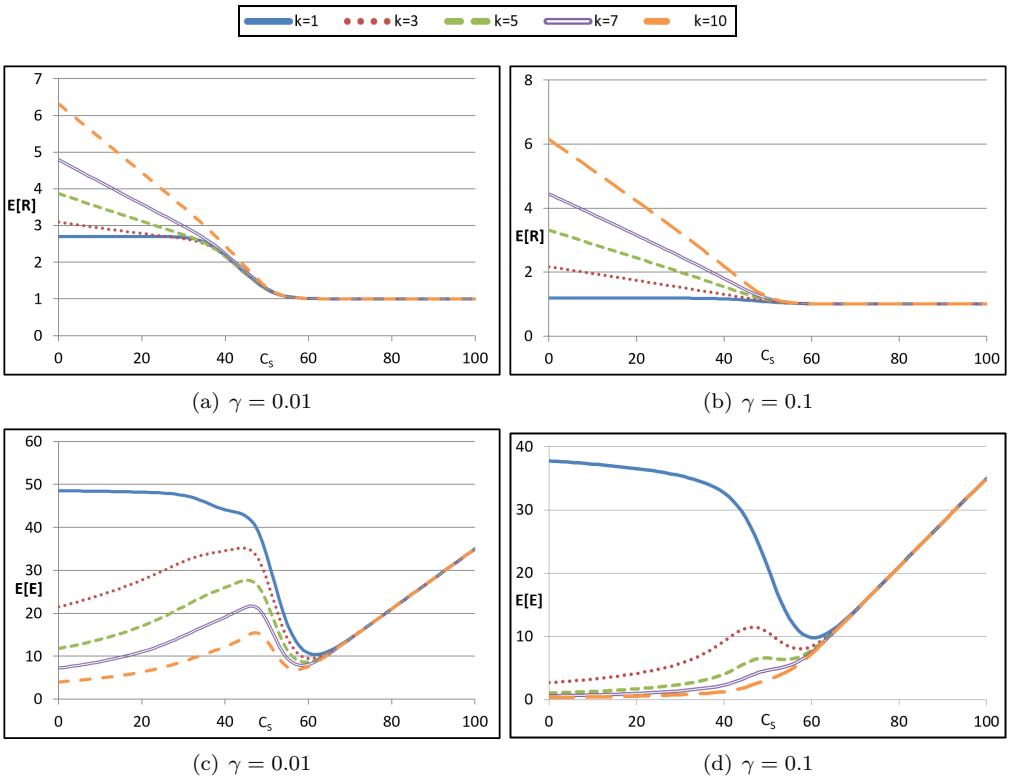
$$T_{i,j} = \frac{1 + f(i,j)\gamma T_{i+1,j} + \lambda T_{i,j+1} + \lambda \sum_{m=i+1}^{C} T_{m,j} P_m(i,j+1)}{\min(i,j)\mu + f(i,j)\gamma + \lambda(1 - P_i(i,j+1))}$$

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

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

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 $(C, (C - C_S + 1)k + C_S - 1)$, then proceed left along that row solving the values for each state, then move down one row to the rightmost state, i.e. $(C-1, (C - C_S + 1)k + C_S - 1)$, and repeat.

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, the source code can be found at [1]. Furthermore, each experiment evaluates the system for every valid value of C_S ($0 \leq C_S \leq C$), while each curve represents a different choice of the threshold value k. As a reminder, C_S denotes the number of static servers, and k is a threshold value defined precisely in the policy definitions; in general, the larger the value of k the more jobs need be present in the system before a server begins its setup process. For all configurations we fix $\mu = 1$ and $\lambda = C/2$. Fixing λ does not significantly limit the overall system behaviours; these systems are designed to dynamically switch servers on and off as needed, and therefore will dynamically adjust the perceived load relatively independent of the load the system would experience if all servers were always on. That is, for a larger system, e.g. $C = 100$, $\mathbb{E}[R]$ would be similar under



(a) $\gamma = 0.01$ (b) $\gamma = 0.1$

(c) $\gamma = 0.01$ (d) $\gamma = 0.1$

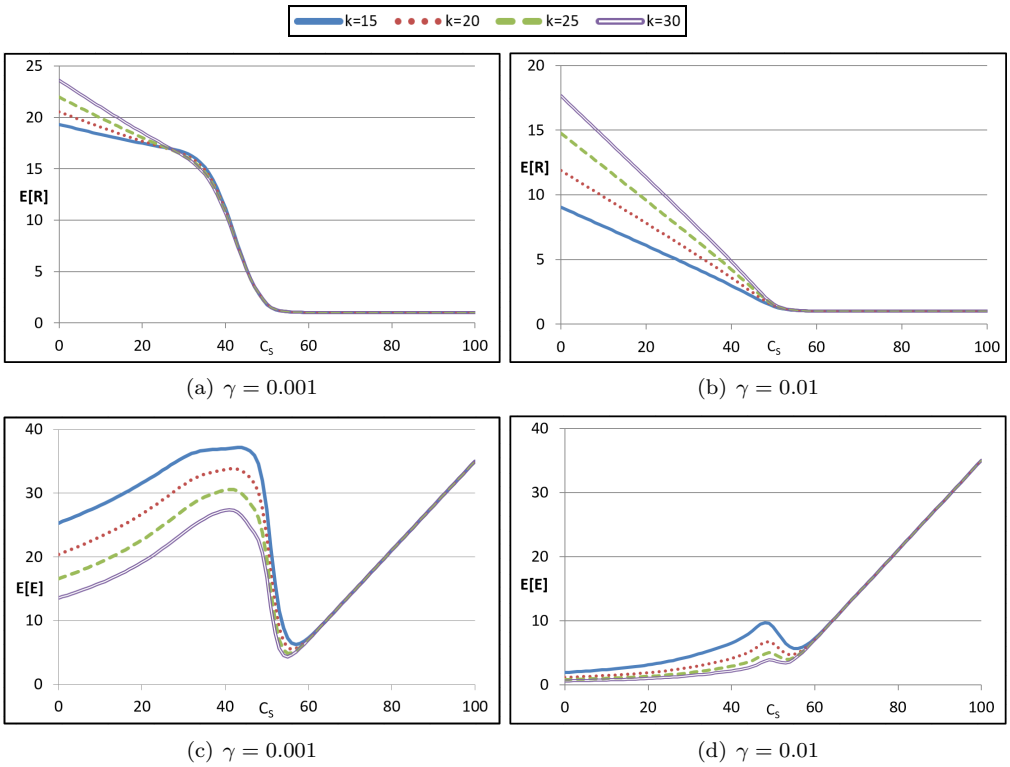
Figure 3: Bulk setup $\mathbb{E}[R]$ vs C_S for (a) and (b), and corresponding $\mathbb{E}[E]$ vs C_S for (c) and (d), $C = 100$, $\lambda = 50$, $\mu = 1$

loads $\rho = 0.25$ and $\rho = 0.5$, since in both cases there is a significant number of servers which are rarely utilized. This insensitivity to the system load can be seen experimentally in [26]. Furthermore, in [23] we demonstrate that as C gets larger, the system behaviour becomes relatively insensitive to it; therefore we also fix $C = 100$. Note that for $C_S = C$, regardless of which policy, or choice of k the system will behave as a static $M/M/C$ queue. An extensive suite of experiments can be found in [23] where some of these conditions are relaxed.

Before we proceed with our results and discussion, it is worth revisiting cost functions which are usually associated with these models. In the literature, different authors use different cost functions. As an example, the authors of [10] focus on the energy response product, i.e. $\mathbb{E}[E]\mathbb{E}[R]$, while others [6, 22] use a linear sum of the metrics, i.e. $\mathbb{E}[R] + \beta\mathbb{E}[E]$ for some $\beta > 0$. As mentioned previously, a 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 well-formed cost functions. 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.

4.4. Bulk Setup - Experiments

We firstly inspect the behaviour of $\mathbb{E}[R]$ under the bulk setup policy. This behaviour can be seen in Figures 3 and 4 (a) and (b). As expected, $\mathbb{E}[R]$ is monotonically decreasing in C_S. However, $\mathbb{E}[R]$ has a more interesting relationship with regards to the choice of k. One might think that a smaller value of k leads to a more proactive system (servers turn on more quickly when queue lengths develop) and thus lower expected response times. However, this is not always the case. Figure 4 (a) is an example of this. Here



(a) $\gamma = 0.001$ (b) $\gamma = 0.01$

(c) $\gamma = 0.001$ (d) $\gamma = 0.01$

Figure 4: Bulk setup with larger k, $\mathbb{E}[R]$ vs C_S for (a) and (b), and corresponding $\mathbb{E}[E]$ vs C_S for (c) and (d), $C = 100$, $\lambda = 50$, $\mu = 1$

for some lower values of C_S (around $30 - 40$) the expected response time for $k = 15$ is actually the largest among all curves shown. While at first perplexing, there is an intuitive explanation. 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, but 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. Figure 4, shows another interesting behaviour. It seems that when k is sufficiently large, the expected response time decreases linearly with C_S until a point where it practically equals $1/\mu$. The point at which this changes in relation to C_S 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 a job has to wait in queue decreases, and its response time approaches its service time. On the other hand, if C_S is lower, the probability of a job 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 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 server increases with C_S. However, the expected time for a server to become available decreases with C_S. 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_S.

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 (c) and (d) of Figures 3 and 4. As a reminder, here $\mathbb{E}[E]$ 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 cost is not monotonically decreasing (nor increasing) in C_S. This leads to local maxima and minima within a given 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 a significant proportion of the static servers to regularly 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 [16]), 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 $C_S = \rho + \sqrt{\rho}$ 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 $C_S = \rho$.*

Looking back at the expected response time, the observation of $C_S = \rho + \sqrt{\rho}$ being a good choice for C_S 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 (a) and (b) of Figures 3 and 4. Furthermore, while $\mathbb{E}[E]$ is sensitive to some configurations around $C_S = \rho + \sqrt{\rho}$, it is less sensitive when C_S is overestimated. In other words, around $C_S = \rho + \sqrt{\rho}$, $\mathbb{E}[E]$ increases at a lower rate when C_S increases, than if C_S were to decrease. This is also good news for system efficacy, as $\mathbb{E}[R]$ is monotonically decreasing in C_S. Therefore, if one wished to err on the side of caution one could set their choice of C_S to be greater than the value which minimizes $\mathbb{E}[E]$ without being punished too harshly.

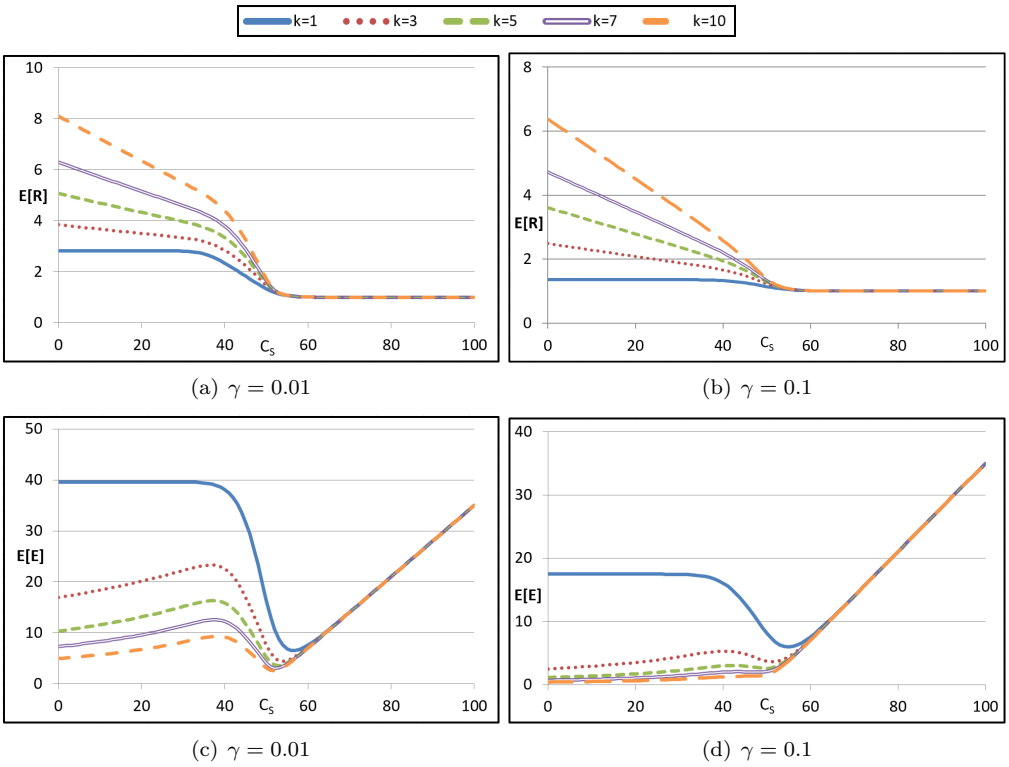
Observation 4. *For low values of γ (longer setup times), the value of C_S which minimizes $\mathbb{E}[E]$ and the value of C_S which minimizes $\mathbb{E}[R]$ are approximately equal.*

4.5. Staggered threshold - Experiments

We complete our numerical results with the *staggered threshold* policy. As discussed previously, this policy aims to capture the simplicity (two decision variables) 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 in Section 4.4. 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 mean response time does not decrease as close to linearly here as it did in the bulk setup results. Figure 6 (b) is a good example of this, in contrast to Figure 4 (b).

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 $\rho + \sqrt{\rho}$, 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 5 (d), while the minimum is actually at $C_S = 0$, the value at $\rho + \sqrt{\rho}$ is still only a slight increase from the minimum value. Therefore, for all the experiments we ran, $\rho + \sqrt{\rho}$ is a reasonable choice for C_S with regards to energy costs as well as system performance. Moreover, inspecting the choice of k for this value of C_S leads to an interesting implication.



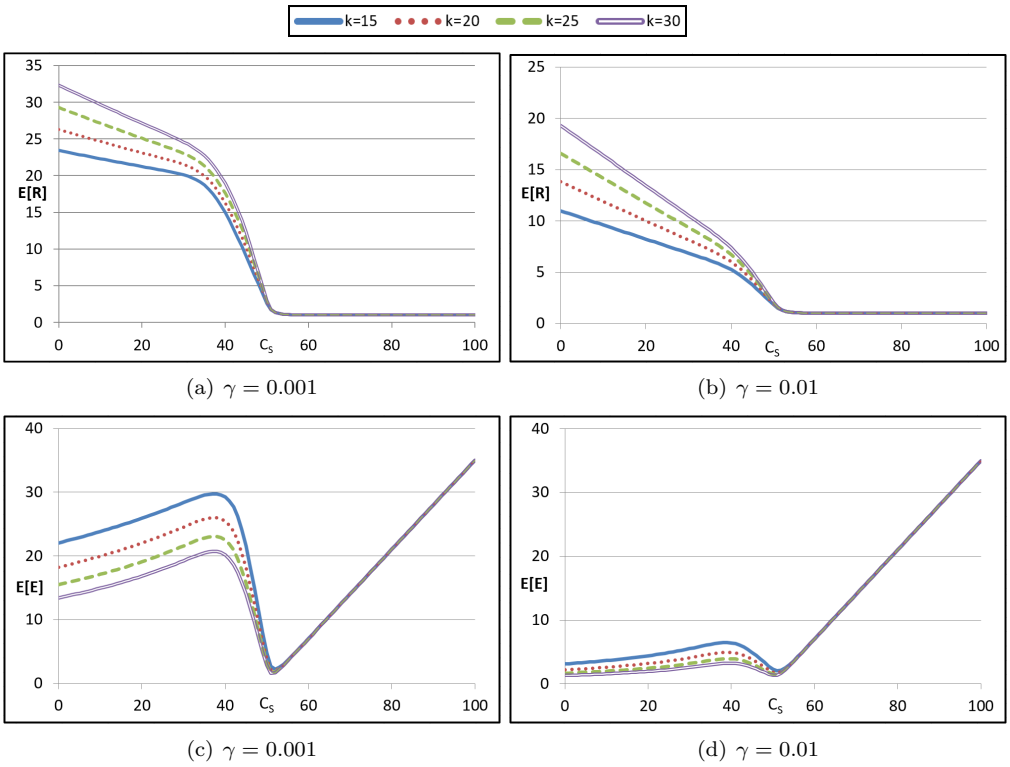
(a) $\gamma = 0.01$ (b) $\gamma = 0.1$

(c) $\gamma = 0.01$ (d) $\gamma = 0.1$

Figure 5: Staggered threshold $\mathbb{E}[R]$ vs C_S for (a), (b) and corresponding $\mathbb{E}[E]$ vs C_S for (c), (d), $C = 100$, $\lambda = 50$, $\mu = 1$

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

Reviewing (c) and (d) in Figures 5 and 6 one will note that for all fixed values of C_S 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 a large number of excess jobs are in the system by the time the next server completes its setup, causing 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 when 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. However, this may not be the case for the choice $C_S = \rho + \sqrt{\rho}$. Viewing Figures 3-6 (a) and (b), one notes that around $C_S = \rho + \sqrt{\rho}$ 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 $M/M/C_S$



(a) $\gamma = 0.001$ (b) $\gamma = 0.01$

(c) $\gamma = 0.001$ (d) $\gamma = 0.01$

Figure 6: Staggered threshold with larger k, $\mathbb{E}[R]$ vs C_S for (a) and (b), and corresponding $\mathbb{E}[E]$ vs C_S for (c) and (d), $C = 100$, $\lambda = 50$, $\mu = 1$

queueing system where $C_S = \rho + \sqrt{\rho}$.

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

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 [24]. 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 $M/M/C_S$ queue is near-optimal for both of these policies. Thus, we argue that an $M/M/C_S$ queue is close to optimal across all potential policies for some C_S. Furthermore, this observation is consistent with the contributions of [10] where they present a similar square root provisioning result for the particular case of the *staggered setup* policy (staggered threshold with $k = 1$ and $C_S = 0$) 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_S. Moreover, the choice of C_S 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 setup and turn off criteria). Such a solution offers an additional benefit. 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 [6, 25, 31]. 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 $M/M/C_S$ queue being a near-optimal solution is further reinforced 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 $M/M/C_S$ queue. That is, $P(N > n) < P(N_{M/M/C_S} > n)$, where $N_{M/M/C_S}$ is a random variable denoting the number of jobs in an $M/M/C_S$ queue, and $C_S < C$. But using $C_S = \lceil \rho + \sqrt{\rho} \rceil = 58$ 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.

This simple solution, while derived from a mathematical model, agrees with practical results from real-world policies. As an example, in [13] the authors note that *dynamic capacity management policies* are too quick to turn idle servers off. Examining Figures 3-6 reinforces this notion. One can quickly observe that not having a reasonable number of static servers (say 40 and below for these particular experiments) results in a severe lose-lose scenario.

5. Conclusion

Provisioning server farms and datacenters is an actively studied and open problem in the intersection of green computing and queueing theory. This work examined an established multiserver queueing model, where each server can be turned on, taking an exponentially distributed amount of time, to improve performance; or turned off instantly to reduce costs. A problem that arises is which policy should be employed for a given cost function, and furthermore, whether this policy will be robust enough to be a reasonable choice when evaluated under other cost functions. To address this issue we set out to draw generic conclusions as opposed to statements which hold only for a specific policy under a specific cost function.

To achieve this we derived several key structural properties which all optimal policies exhibit under linear cost functions. Specifically, the threshold nature of setups and turn offs, turning a server off if there is a job which it could be processing, and the bulk setup nature of the setups. Using this information, one can build confidence in previously analysed policies which adhere to these properties. Moreover, when constructing new policies to consider, one can leverage these structural properties and use them as formal justification for choices made rather than relying solely on intuition.

With this in mind we presented two specific policies to analyse further, i.e. *bulk setup*, and *staggered threshold*, which adhere to the aforementioned structural properties. 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 cost for both 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 observations which grant insight into how these systems behave. This includes, but is not limited to, our argued degenerative solution that an $M/M/C_S$ queue is reasonably close to optimal across all potential policies and cost functions for some choice of C_S around $\rho + \sqrt{\rho}$.

Looking forward to the future of this work there is no shortage of possible alterations or extensions to the model. For example, how well do these policies fare under a time varying arrival rate? It is our intuition that if the arrival rate varies on a relatively long time scale, with regards to other system parameters such as the expected setup time, then the results given here may be reasonable to apply, where C_S is dynamically chosen over an appropriate time window. The mechanics of the model could also be changed to perhaps better fit reality, such as allowing servers to be heterogeneous. Here certain servers could process jobs at a higher rate, but at a greater energy cost. We surmise that if the energy costs increased linearly with the processing rate then our observations here could be easily extended. However, when energy costs increase non-linearly with the processing rate (as is more realistic) this problem becomes much more interesting. One last example direction would be relaxing the assumption of exponentially distributed processing times. Our intuition

is that the observation of a simple static provisioning based on the means of the underlying distributions would be reasonable, but one could run simulations (possibly incorporating real world traces) to verify this. While the above considerations are certainly deserving of attention, the analysis presented here allows one to confidently make generic statements and conclusions across the problem domain, specifically, how many servers should be statically provisioned.

Acknowledgement. This research was funded by the Natural Sciences and Engineering Research Council of Canada. The authors would also like to acknowledge the two reviewers of this paper, as their extremely diligent and insightful critiques allowed us to significantly strengthen this work.

[1] Source code. `http://www.cas.mcmaster.ca/~macciov/publications.html`. Accessed: 2016-10-10.

[2] A. Allahverdi, C. Ng, T. Cheng, and M. Y. Kovalyovc. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

[3] J. R. Artalejo. A unified cost function for M/G/1 queueing systems with removable server. *Trabajos de Investigacion Operativa*, 7(1):95–104, 1992.

[4] L. A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

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

[6] 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*, 33(1):303–314, June 2005.

[7] S. Doroudi, B. Fralix, and M. Harchol-Balter. Clearing analysis on phases: Exact limiting probabilities for skip-free, unidirectional, quasi-birth-death processes. *Stochastic Systems*, 6(2):420–458, 2016.

[8] EPA. Report to congress on server and data center energy efficiency. Technical report, U.S Environmental Protection Agency, 2007.

[9] 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*, pages 153–166, 2013.

[10] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, 2010.

[11] A. Gandhi and M. Harchol-Balter. M/M/k with exponential setup. Technical report, Carnegie Mellon University, 2010.

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

[13] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Transactions on Computer Systems*, 30(4):14:1–14:26, Nov. 2012.

[14] 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*, pages 82–89, 2014.

[15] M. E. Gebrehiwot, S. Aalto, and P. Lassila. Energy–performance trade-off for processor sharing queues with setup delay. *Operations Research Letters*, 44(1):101–106, 2016.

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

[17] J. Hu and T. Phung-Duc. Power consumption analysis for data centers with independent setup times and threshold controls. In *AIP*, 2015.

[18] 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)*, pages 1–9, 2014.

[19] 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*, 75–76:17–35, 2014.

[20] 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.

[21] 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*, 25(2):497–504, 2015.

[22] 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*, pages 233–244, 2011.

[23] V. J. Maccio and D. G. Down. Exact analysis of energy-aware multiserver queueing systems with setup times. Technical Report CAS-16-01-DD, Department of Computing and Software, McMaster University.

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

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

[26] V. J. Maccio and D. G. Down. Asymptotic performance of energy-aware multiserver queueing systems with setup times. Technical report, McMaster University, 2016.

[27] V. J. Maccio and D. G. Down. Exact analysis of energy-aware multiserver queueing systems with setup times. In *Proceedings of the IEEE 24st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2016)*, pages 11–20, September 2016.

[28] M. J. Magazine. On optimal control of multi-channel service systems. *Naval Research Logistics Quarterly*, 18(2):429–441, 1971.

[29] M. Mazzucco and D. Dyachuk. Optimizing cloud providers revenues via energy efficient server allocation. *Sustainable Computing: Informatics and Systems*, 2(1):1–12, 2012.
[30] I. Mitrani. Managing performance and power consumption in a server farm. *Annals of Operations Research*, 202(1):121–134, 2013.
[31] T. H. Nguyen, M. Forshaw, and N. Thomas. Operating policies for energy efficient dynamic server allocation. *Electronic Notes in Theoretical Computer Science*, 318:159–177, 2015.
[32] 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.
[33] T. Phung-Duc. Exact solutions for M/M/c/setup queues. *Telecommunication Systems*, pages 1–16, 2016.
[34] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 2005.
[35] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the electric bill for internet-scale systems. 39(4):123–134, 2009.
[36] Y. Ren, T. Phung-Duc, Z. W. Yu, and J. C. Chen. Design and analysis of dynamic auto scaling algorithm (DASA) for 5G mobile networks. *arXiv preprint arXiv:1604.05803*, 2016.
[37] 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*, pages 247–261, 2008.
[38] N. Tian and Z. G. Zhang. A two threshold vacation policy in multiserver queueing systems. *European Journal of Operational Research*, 168(1):153–163, 2006.
[39] N. Tian and Z. G. Zhang. *Vacation Queueing Models - Theory and Applications*. Springer Science, 2006.
[40] X. Xu and N. Tian. The M/M/c queue with (e, d) setup time. *Journal of Systems Science and Complexity*, 21(3):446–455, 2008.
[41] 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.

Appendix A. Proofs

The proofs of Theorems 1, 2, 3 introduced in Section 3 are presented here. Before this is done, we present a Markov decision process (MDP) formulation of an energy-aware system $S = (C, \lambda, \mu, \gamma)$. Without loss of generality we assume S has undergone uniformization, i.e. $\lambda + C\max(\mu, \gamma) = 1$. We also use the notation from [34] where A_s denotes the allowable set of actions in state s, and $p(s'|s, a)$ denotes the probability of being in state s' at the next decision epoch, given that at the current decision epoch the system is in state s and performs action a. An action a is the number of servers that will be in setup until the next decision epoch if $a > 0$. If $a < 0$ this denotes the number of on servers which will be turned off (it is also assumed that no servers will be in setup until the next decision epoch). For the MDP, similar to the CTMC, a state is given by (i, j), where i denotes the number of servers on and j denotes the number of jobs in the system. With uniformization and the above notation in mind, an MDP for an energy-aware system can be defined by the following equations,

$$p((i',j')|(i,j),a) = \begin{cases} \lambda & \text{if} \quad j' = j+1,\ i' = i + \min(0,a) \\ \min(j, i+\min(0,a))\mu & \text{if} \quad j' = j-1,\ i' = i + \min(0,a) \\ \max(0,a)\gamma & \text{if} \quad j' = j,\ i' = i+1 \\ 1 - \lambda \\ -\min(j, i+\min(0,a))\mu \\ \quad -\max(0,a)\gamma & \text{if} \quad j' = j,\ i' = i + \min(0,a) \end{cases}$$

and

$$A_{(i,j)} = \{-i, -i+1, ..., -1, 0, 1, ..., C-i-1, C-i\},$$

and the cost (reward) function $w((i, j), a)$, which for now is left unspecified. Note that if one wished to *continue* the setup processes of all servers from the previous decision epoch, then a would have to be chosen

as if said setup processes were just starting at the current decision epoch. Due to the Markovian nature of the dynamics, these two viewpoints are mathematically equivalent. From here one can define the usual optimality equations as follows

$$\begin{aligned} c^* + u(i,j) = \min_{a \in A_{(i,j)}} \{ & w((i,j),a) + \lambda u(i + \min(0,a), j+1) + (\max(0,a))\gamma u(i+1,j) \\ & + \min(j, (i + \min(0,a)))\mu u(i + \min(0,a), j-1) \\ & + (1 - \lambda - \max(0,a)\gamma - \min(j,(i+\min(0,a)))\mu) u(i+\min(0,a), j)\}, \end{aligned}$$

where c^* is the optimal average cost.

In addition to the traditional optimality equations, we find that it is useful to also define a set of *constrained optimality equations* defined as

$$\begin{aligned} c^* + u(i,j,a) = & \, w((i,j),a) + \lambda u(i + \min(0,a), j+1) + \max(0,a)\gamma u(i+1,j) \\ & + \min(j, (i + \min(0,a)))\mu u(i + \min(0,a), j-1) \\ & + (1 - \lambda - \max(0,a)\gamma - \min(j,(i+\min(0,a)))\mu) u(i+\min(0,a), j). \end{aligned}$$

These equations follow the optimal actions (determined by the traditional optimality equations), except at the first decision epoch, where the action is explicitly specified. It will be seen that a proof of one of the main results is aided through the use of these equations.

For the sake of completeness the general threshold policy, the bulk setup, and the staggered threshold policies are defined in the context of the MDP. That is, once the parameters of each policy are fixed, then for each state (i,j) the action in that state denoted by $a_{i,j}$ is uniquely determined. For the general threshold policy, for all $m < C$,

$$a_{i,j} = \begin{cases} -i & j \le k_1^- \\ m - i - 1 & k_{m-1}^- \le j \le k_m^- \\ 0 & k_i^- < j < k_{i+1,i}^+ \\ m - i & k_{m,i}^+ \le j < k_{m+1,i}^+ \\ C - i & k_{C,i}^+ \le j \end{cases}$$

For the bulk setup policy with instantiated decision variables C_S and k the action in each state is:

$$a_{i,j} = \begin{cases} j - i & C_S \le j \le i \\ C - i & C_S + (i - C_S + 1)k \le j \\ 0 & \text{otherwise.} \end{cases}$$

For a staggered threshold policy with instantiated decision variables C_S and k the action in each state is

$$a_{i,j} = \begin{cases} j - i & C_S \le j \le i \\ \min(C - i, \lfloor (j - C_S)/k \rfloor - i + C_S)) & C_S + (i - C_S + 1)k \le j. \\ 0 & \text{otherwise.} \end{cases}$$

With these definitions in hand, we proceed with the proofs of the theorems presented in Section 3.

Appendix A.1. Proof of Theorem 1

For simplicity of navigation Theorem 1 is restated.

Theorem 1. *For all energy-aware systems, for all linear well-formed cost functions, the optimal policy is a threshold policy.*

Proof. To prove that the optimal policy is a threshold policy three properties must be shown. Firstly, optimal actions are always made the moment an event occurs. Secondly, any benefit of turning off the ith server in state $(i, j+1)$ is also gained by turning off the ith server in state (i, j). Lastly, any benefit gained by turning on a server(s) in state (i, j), is also gained by turning on a server(s) in state $(i, j+1)$.

The first property is an immediate consequence of all underlying distributions being exponential. As such, the proof moves to the second property (the threshold nature of the turn offs). This is shown via contraposition. That is, if it is optimal to keep the ith server on in state (i, j), then it is also optimal to keep the ith server on in state $(i, j+1)$, if and only if the second property holds.

Therefore, we show that any benefit from keeping the ith server on in state (i, j) will also occur in state $(i, j+1)$. In fact, keeping the ith server on can be even more beneficial in state $(i, j+1)$. Firstly, if the ith server could process a job in state (i, j), then it can clearly process a job in state $(i, j+1)$ as there are more jobs in the system. Secondly, if the server were to idle in anticipation of a job arriving in state (i, j), then it could also idle in state $(i, j+1)$, with an even higher probability that it will be utilized in the future since there are more jobs present. Lastly, there exists the case of $j = i - 1$, where in state (i, j) the server would idle, while in state $(i, j+1)$ it would be able to process jobs. If however, for whatever reason in state $(i, j+1)$ it would instead be better for the system for the ith server to idle rather than process the job (to mimic its behaviour in state (i, j)) this would also be an option. Therefore, any benefit (lesser cost incurred) which keeping on the ith server on offers in state (i, j) is also present in state $(i, j+1)$.

One may note that above we mention the case of letting a server idle while there is a job waiting in queue, while the MDP defined at the beginning on this appendix does not allow for such behaviour. This is easily rectified however, by noting that such behaviour is always suboptimal. For example, if a server could be processing a job, but idles for an interval of length t before it (or another server) begins processing said job, it would always be better to instead process the job and then idle for an interval of length t since the same energy cost is incurred while the holding cost is decreased.

To show that the turn on criteria also follows a threshold policy is slightly more complex, since unlike turn offs, setups do not happen instantaneously. We first show a consequence of the optimality equations. Before this is done, an observation is made regarding the MDP's immediate cost function. For all stable policies, $\mathbb{E}[E]$ has a component $E_{\text{Busy}}\lambda/\mu$. As a result, a linear well-formed cost function, i.e. $\mathbb{E}[R] + \beta\mathbb{E}[E]$, is minimized when one employs the optimal policy determined by the immediate cost function

$$w((i, j), a) = j/\lambda + \beta[\max(0, (i - j + \min(0, a)))E_{\text{Idle}} + \max(0, a)E_{\text{Setup}}].$$

As such, the above immediate cost function is assumed to be used for the remainder of this proof.

$$\begin{aligned}
&u((i, j), 1) \leq u((i, j), 0) \\
\Leftrightarrow\ &\max(i - j, 0)\beta E_{\text{Idle}} + \beta E_{\text{Setup}} + \lambda u(i, j+1) + \min(i, j)\mu u(i, j-1) + \gamma u(i+1, j) \\
&\quad + [1 - \lambda - \min(i, j)\mu - \gamma]u(i, j) \\
&\quad \leq \max(i - j, 0)\beta E_{\text{Idle}} + \lambda u(i, j+1) + \min(i, j)\mu u(i, j-1) + [1 - \lambda - \min(i, j)\mu]u(i, j) \\
\Leftrightarrow\ &\beta E_{\text{Setup}} \leq \gamma u(i, j) - \gamma u(i+1, j) \\
\Leftrightarrow\ &\frac{\beta E_{\text{Setup}}}{\gamma} \leq u(i, j) - u(i+1, j) \qquad \text{(A.1)}
\end{aligned}$$

This implies it is optimal to turn a server on in state (i, j) if and only if the cost saved by having an extra server on, thus bringing it to state $(i+1, j)$ is greater than or equal to the expected cost of turning a server on. Therefore, if it is shown

$$u(i, j) - u(i+1, j) \leq u(i, j+1) - u(i+1, j+1) \qquad \text{(A.2)}$$

then the optimal policy is a threshold policy (regarding setups). In other words, if having an extra server on in state $u(i, j+1)$ can save just as much cost of having an extra server on in state $u(i, j)$ then the turn on criteria follows a threshold policy. Note that reasoning about these systems as seen in (A.2) is convenient since in states $(i+1, j)$ and $(i+1, j+1)$ the *extra* server is already on, and the setup times can be disregarded. The inequality (A.2) is shown by noting that whenever an extra server in $u(i, j)$ could save cost, so could an extra server in $u(i, j+1)$.

Let s_1 and s_2 be extra servers in states (i, j) and $(i, j+1)$ respectively, i.e. utilizing s_1 and s_2 would bring the systems to states $(i+1, j)$ and $(i+1, j+1)$, respectively. Whatever action s_1 performs, s_2 can mimic it. That is, if s_1 processes a job, idles, or turns off, s_2 can follow suit. Moreover, there exists the case where $i = j$ in which if s_1 idles, s_2 can instead process a job. Thus in this case, s_2 actually saves greater cost than s_1. Therefore, s_2 can save the system just as much cost as s_1. Then if it is optimal to turn s_1 on in state (i, j), then from (A.1) it follows s_1 can save cost of at least $\beta E_{\text{Setup}}/\gamma$. Therefore s_2 can save cost of at least $\beta E_{\text{Setup}}/\gamma$ which again from (A.1) implies it is optimal to turn a server on in state $(i, j+1)$. □

Appendix A.2. Proof of Theorem 2

For simplicity of navigation Theorem 2 is restated.

Theorem 2. *For all energy-aware systems, for all cost functions of the form*

$$\mathbb{E}[R] + \beta\mathbb{E}[E],$$

where $\beta\lambda E_{Idle} < 1$, if the number of jobs in the system is greater than or equal to the number of servers currently turned on, it is always suboptimal to turn a server off, which is already on.

Proof. The proof of this theorem is achieved via noting a contradiction in the optimality equations of the MDP if the negation of the theorem is assumed to be true. As noted previously, for all stable policies, $\mathbb{E}[E]$ has a component $E_{\text{Busy}}\lambda/\mu$. As a result, a linear well-formed cost function, i.e. $\mathbb{E}[R] + \beta\mathbb{E}[E]$, is minimized when one employs the optimal policy determined by the immediate cost function

$$w((i, j), a) = j/\lambda + \beta[\max(0, (i - j + \min(0, a)))E_{\text{Idle}} + \max(0, a)E_{\text{Setup}}].$$

As such, the above immediate cost function is assumed to be used for the remainder of this proof. Assume that $j \geq i$ (there are at least as many jobs in the system as there are servers on), and that it is better to turn a server off, rather than keep all of them on. From the optimality equations, this implies that $u(i, j, -1) < u(i, j, 0)$.

$$\begin{aligned}
& u(i,j,-1) < u(i,j,0) \\
\Leftrightarrow\ & w((i,j),-1) \\
& \quad + \lambda u(i-1,j+1) + (i-1)\mu u(i-1,j-1) + [1-\lambda-(i-1)\mu]u(i-1,j) \\
& \quad < w((i,j),0) + \lambda u(i,j+1) + i\mu u(i,j-1) + [1-\lambda-i\mu]u(i,j) \\
\Leftrightarrow\ & j + \lambda u(i-1,j+1) + (i-1)\mu u(i-1,j-1) + [1-\lambda-(i-1)\mu]u(i-1,j) \\
& \quad < j + \lambda u(i,j+1) + i\mu u(i,j-1) + [1-\lambda-i\mu]u(i,j) \\
\Leftrightarrow\ & \lambda u(i-1,j+1) + (i-1)\mu u(i-1,j-1) + [1-\lambda-(i-1)\mu]u(i-1,j) \\
& \quad < \lambda u(i,j+1) + i\mu u(i,j-1) + [1-\lambda-i\mu]u(i,j) \qquad \text{(A.3)}
\end{aligned}$$

It is observed that for all n and m, $u(n, m) \leq u(n-1, m)$. This follows from noting that in state (n, m) the system could always immediately turn a server off bringing it to state $(n-1, m)$ without additional cost. Therefore, (A.3) implies

$$[1-\lambda-(i-1)\mu]u(i-1,j) \quad < \quad [1-\lambda-i\mu]u(i,j) + \mu u(i,j-1). \qquad \text{(A.4)}$$

While it is true that for all n and m, $u(n,m) \leq u(n-1,m)$ something even stronger can be said regarding $u(i,j)$ and $u(i-1,j)$. Because it is assumed that it is better to turn a server off in state (i,j) and that servers can turn off instantly it is known $u(i,j) = u(i-1,j)$. Therefore, (A.4) implies

$$\begin{aligned} [1-\lambda-(i-1)\mu]u(i,j) &< [1-\lambda-i\mu]u(i,j)+\mu u(i,j-1) \\ -(i-1)\mu u(i,j) &< -i\mu u(i,j)+\mu u(i,j-1) \\ \mu u(i,j) &< \mu u(i,j-1) \\ u(i,j) &< u(i,j-1). \end{aligned} \tag{A.5}$$

Now let the ith server in the system with $j-1$ jobs be denoted by s_1 and the ith server in the system with j jobs be denoted by s_2. Consider the sample path of s_1 mimicking s_2 with the exception of s_1 idling while s_2 is busy when there is no job for s_1 to serve. Let this mimicking continue until the two systems become synchronized. Assuming $\beta E_{\text{Idle}} < 1/\lambda$ (or $\beta\lambda E_{\text{Idle}} < 1$) implies the system with one less job will incur a lesser or equal cost. Therefore, $u(i,j-1) \leq u(i,j)$ which contradicts (A.5). It is worth noting that while the assumption $\beta\lambda E_{\text{Idle}} < 1$ is required for the proof, we conjecture that the theorem holds under more general conditions.

□

Appendix A.3. Proof of Theorem 3

For ease of reading Theorem 3 is restated.

Theorem 3. *For all energy-aware systems, for all linear well-formed cost functions the optimal policy turns servers on following a bulk setup scheme.*

Proof. As noted previously, for all stable policies, $\mathbb{E}[E]$ has a component $E_{\text{Busy}}\lambda/\mu$. As a result, a linear well-formed cost function, i.e. $\mathbb{E}[R] + \beta\mathbb{E}[E]$, is minimized when one employs the optimal policy determined by the immediate cost function

$$w((i,j),a) = j/\lambda + \beta[\max(0,(i-j+\min(0,a)))E_{\text{Idle}} + \max(0,a)E_{\text{Setup}}].$$

As such, the above immediate cost function is assumed to be used for the remainder of this proof. The strategy is to leverage the optimality equations of the MDP to show the bulk setup nature of the optimal policy. It is assumed that in state (i,j) it is better to have a servers in setup, where $a > 0$, rather than choose to do nothing. The following inequality is then known from the optimality equations.

$$\begin{aligned} & u(i,j,a) \leq u(i,j,0) \\ \Leftrightarrow\quad & w((i,j),a) + \lambda u(i,j+1) + a\gamma u(i+1,j) + \min(i,j)\mu u(i,j-1) + [1-\lambda-a\gamma-\min(i,j)\mu]u(i,j) \\ & \quad \leq w((i,j),0) + \lambda u(i,j+1) + \min(i,j)\mu u(i,j-1) + [1-\lambda-\min(i,j)\mu]u(i,j) \\ \Leftrightarrow\quad & a\beta E_{\text{Setup}} + a\gamma u(i+1,j) - a\gamma u(i,j) \leq 0 \\ \Leftrightarrow\quad & \frac{\beta E_{\text{Setup}}}{\gamma} \leq u(i,j) - u(i+1,j) \end{aligned}$$

This derivation can be expressed more succinctly as

$$u(i,j,a) \leq u(i,j,0) \quad \Leftrightarrow \quad \frac{\beta E_{\text{Setup}}}{\gamma} \leq [u(i,j) - u(i+1,j)], \tag{A.6}$$

for all valid actions $a \geq 0$. Now consider the action of turning on all available servers. If it can be shown that this option is at least as good as turning on an arbitrary choice of a (when $a \geq 0$), then the proof is complete. This is expressed mathematically as follows.

$$u(i,j,C-i) \leq u(i,j,a)$$

$$\begin{aligned}\Leftrightarrow \quad & w(i,j,C-i) + \lambda u(i,j+1) + (C-i)\gamma u(i+1,j) + \min(i,j)\mu u(i,j-1) \\ & + [1-\lambda-(C-i)\gamma - \min(i,j)\mu]u(i,j) \\ & \quad \leq w(i,j,a) + \lambda u(i,j+1) + a\gamma u(i+1,j) + \min(i,j)\mu u(i,j-1) + [1-\lambda-a\gamma-\min(i,j)\mu]u(i,j)\end{aligned}$$

$$\Leftrightarrow \quad (C-i-a)\beta E_{\text{Setup}} + (C-i-a)\gamma u(i+1,j) \leq (C-i-a)\gamma u(i,j)$$

$$\Leftrightarrow \quad \frac{\beta E_{\text{Setup}}}{\gamma} \leq u(i,j) - u(i+1,j)$$

This derivation can be expressed more succinctly as

$$u(i,j,C-i) \leq u(i,j,a) \quad \Leftrightarrow \quad \frac{\beta E_{\text{Setup}}}{\gamma} \leq [u(i,j) - u(i+1,j)],$$

for all valid actions a. Then from (A.6)

$$u(i,j,C-i) \leq u(i,j,a) \quad \Leftrightarrow \quad u(i,j,a) \leq u(i,j,0),$$

for all valid actions $a \geq 0$. Or in other words, if it is optimal to turn at least one server on, then it is optimal to turn all available servers on. Hence it is optimal to turn servers on following a *bulk setup* scheme.

□