Stability of Parallel-Server Systems with Service Elasticity

Misha Wei Sohan

April 2019

Undergraduate Honors Thesis
Applied Mathematics-Economics
Brown University

Faculty Advisor: Professor Kavita Ramanan
Post-doctoral Advisor: Professor Debankur Mukherjee
Second Faculty Reader: Professor Matthew Harrison

Abstract

Large-scale cloud networks and data centers operate in distributed parallel queue systems, rather than maintaining a centralized queue. Since demand patterns vary depending on time and cannot be predicted with certainty, large-scale cloud networks and data centers face the serious issue of achieving efficient server utilization and energy consumption whilst minimizing user-perceived delay. Achieving optimal service elasticity especially without global queue length information is very challenging - turning off idle servers saves energy, but turning them on when needed requires a long setup period, which typically exceeds the latency tolerance of real-time processing and control functions by orders-of-magnitude.

Recently, a token-based joint auto-scaling and load balancing strategy that maintains constant communication overhead has been proposed in the literature for such systems. The scheme achieves certain asymptotic optimality in terms of delay performance and energy consumption when the number of servers, N, tends to infinity. However, for a fixed finite value of N, it is not clear that the system under the proposed scheme is even stable. In fact, for some choice of parameters it is known that the system is unstable for small N. Our primary goal in this project is to investigate this stability issue for finite N-values.

Specifically, we propose an alternative scheme that provides better service elasticity and robustness for systems with finite N, whilst still maintaining constant communication overhead. We prove the stability of the proposed scheme and corroborate these theoretical results using simulations. Using additional simulations, we also investigate the scheme's performance in terms of average expected wait time and average expected energy consumption compared to the existing token-based joint auto-scaling and load balancing scheme. The results show that the proposed scheme performs better in both aspects.

Contents

1 Introduction

Background and motivation.
In recent years data has become increasingly abundant and valuable as daily life in today's information age is interconnected with the digital world. It was found that an estimated 2.5 quintillion bytes of data is generated daily. This has necessarily led to a rise in demand for data storage. More and more applications are being hosted in both traditional data centers and cloud networks. Since 2016 there has been a notable growth in the popularity of cloud networks due to its scalability and the ease of outsourcing to third-party cloud providers. In addition, edge computing is also on the rise. With edge computing, compute power is located on the edge of cloud networks, allowing data processing to occur closer to the source of the data. This eliminates the need to stream data to a centralized cloud, thus reducing inefficiencies in network infrastructure. This growing demand for data centers has led to two pertinent issues: the high energy consumption of data centers and the high financial costs associated with maintaining them. In 2014, U.S. data centers used approximately 70 billion kWh, equivalent to roughly 2% of the entire U.S. electricity consumption [1]. It is projected that this figure will grow to 73 billion kWh by 2020 [1] and associated with this are the increased environmental costs. In 2017, data centers across the globe used approximately 420 tWh, contributing to roughly 3% of world electricity consumption and 2% of greenhouse gas emissions [2]. With global data traffic increasing by more than twofold every four years [3], these figures will rise further. Data centers are also expensive to run - a center with 50,000 servers has an estimated cost of $52.5 million per year [4]. The cost of electricity to power the data centers is the largest operating expenditure at 40% to 80% of total expenditure [5]. A large contributor to the high environmental and financial costs is the low server utilization rate which runs as low as 10% [4] - server utilization rate refers to the percentage of switched-on servers in a data center that are actually busy servicing tasks. It is common practice to leave servers running even when not in use so as to ensure the best possible user-perceived performance. Ultra low-latency and high reliability are priorities since small delays can severely impact e-commerce sales and real-time processes [4]. Data center operators conventionally require that the 95^{th} percentile of response times remains below a given threshold. However, the problem is that servers that are left idle on still consume at least 60% of peak power. The total reported annual cost of servers that are left idle on is over $19 billion and they emit more than 11 million tons of carbon dioxide a year [6].

This effects the task of achieving efficient server utilization while still maintaining excellent user-facing performance in the face of varying and unpredictable demand. Surges in traffic for cloud services are commonplace, especially for new services, far in excess of typical forecasts. The capability to dynamically increase and decrease resources in terms of the number of switched-on servers to match the amount of resources needed is key to reducing the environmental and financial costs outlined above. This resonates with the concept of service elasticity in cloud computing. Though turning off idle servers reduces energy wastage and increases utilization rates at that moment, the time required to turn a server back on when needed is significantly higher than the tolerance for delays in executing tasks. This is a key obstacle to achieving optimal service elasticity. Preventing this delay by keeping most or all idle servers on would not solve the issue of the high environmental and financial costs. In addition, the process of turning a server back on uses peak levels of power consumption [7].

One approach employed by large companies in the space is auto-scaling. For example, Amazon Web Services (AWS) has an AWS Auto Scaling offering and Microsoft offers Microsoft Azure Autoscale to dynamically scale apps to respond to changing demand. Most auto-scaling schemes use real-time information from a centralized queue to dynamically turn servers on and off based on the state of the servers [8]. However, in cloud networks and conventional data centers tasks are distributed amongst parallel queues upon arrival, without maintaining a centralized queue. Thus, implementing auto-scaling techniques that are intended for a centralized queue system and which rely on global centralized queue data will result in massive communication overhead. Furthermore, even if the global queue data were to be collected with the communication overhead, it is uncertain how effective auto-scaling techniques would be since, in the parallel queue system, tasks could be waiting at some servers while other servers remain idle.

To target this issue, TABS (Token-based Auto Balance Scaling) - a token-based joint auto-scaling and load balancing strategy for parallel queue server systems - was recently proposed in literature [8]. TABS utilizes a token-based feedback protocol that enables the dispatcher to know which servers are on and busy, on and idle, off, or in setup mode as they are turning on. Without requiring global queue length information, this scheme achieves certain asymptotic optimality in terms of delay performance and energy consumption as the number of servers in the system tends to infinity.

However, for a fixed finite number of servers, it is unclear that the system will be stable under the proposed TABS scheme [9]. Edge data centers are smaller data centers that reach the edge of the cloud network in order to provide compute power to local users more efficiently. With the growing traction of edge computing, edge data centers are becoming increasingly in demand. Correspondingly, more companies are using small, local data centers with far fewer servers than their larger industry player counterparts such as AWS. TABS may not be a reliable and robust scheme for such systems.

2 Literature Review

There is extensive literature on centralized queuing systems. [10] investigates whether it is advantageous, in terms of wait times and energy consumption, to turn servers off when they are idle on in a centralized queue setting. The main policies discussed are the ON/IDLE, ON/OFF and ON/OFF/STAG. The ON/IDLE scheme is essentially the *AlwaysOn* scheme in which servers are never turned off. The ON/OFF scheme turns servers off immediately when they become idle. The ON/OFF/STAG scheme turns servers off when they become idle but allows only one server to be in setup mode at any given time, similar to a staggered setup scheme in an attempt to reduce energy wastage caused by servers in setup mode. Results show that for low setup times, the ON/OFF/STAG scheme has slightly lower power consumption than the ON/OFF policy, but much higher waiting times for tasks. When setup times are high, the ON/IDLE scheme performs better than the ON/OFF scheme when the number of tasks in the system is high. However, when the number is low, the ON/OFF scheme has better results with respect to expected energy consumption and the ON/IDLE scheme performs better in terms of expected waiting time. In response to this, a modified scheme, the ON/IDLE(t) policy, is suggested which mixes the ON/IDLE and ON/OFF policies. Under this policy, an idle server is only turned off if the total number of on servers is above a specified threshold value, t. It is shown that the expected wait time for the ON/IDLE(t^*) scheme is comparable to that of the ON/IDLE scheme and that the expected power consumption is as low as that of the better of the ON/OFF and the ON/IDLE schemes. t^* is defined as the value of t for which the ON/IDLE(t) has the lowest energy usage. However determining t^* in real-time is not feasible.

Another scheme that is of particular relevance is the M/M/N/setup/delayed-off scheme which is a variation of the M/M/N queuing scheme, where N is the number of servers in the system. Under this scheme, servers that are left idle on for an exponentially distributed amount of time - the standby period - are turned off, thus reducing the number of servers left idle on. Idle off servers are turned on, with a setup time that is assumed to be exponentially distributed with a mean of 100 seconds, if a task arrives and there are no idle on servers. In [11], the system is modelled as a Markov Chain and the first exact, closed-form analysis of the scheme is provided. For small job sizes which can be serviced more quickly, it is shown that the M/M/N/setup/delayedoff scheme performs better than other variations of M/M/N queueing schemes in terms of mean wait time of tasks and mean energy consumption. The lower energy consumption is due to the delay before turning the server off, which avoids the high energy cost of turning servers off then on unnecessarily.

Alternative schemes in which idle on servers are put in sleep mode are discussed in [7]. This paper details the energy wastage of the *AlwaysOn* scheme that is the standard in the industry and investigates dynamic power management using sleep states with non-zero setup times. The *AlwaysOn* scheme keeps a fixed number of servers on at all times. The proposed *Reactive* scheme turns servers to sleep mode when the load decreases and turns them back on when the load increases. It is found to perform better than the *AlwaysOn* scheme for only low enough values of setup time and power consumed in sleep state. Its poor performance is due in part to the fact that servers are turned off frequently, which later take a long time to turn on. The *SoftReactive* scheme which turns servers to sleep mode only after waiting a specified amount of time is found to work better. Tasks are sent to the lowest-indexed server that is serving fewer than p tasks, where p is the optimal number of tasks a server can serve concurrently. This method concentrates incoming tasks on a smaller subset of the servers, allowing the remaining servers to timeout and be put to sleep state to conserve energy. In the event that all servers are already serving p requests, new tasks are sent to servers using the Join-the-Idle-Queue (JIQ) policy. Findings show that as the number of servers increases, the setup time affects the performance of the schemes less and thus longer setup times have less of a negative effect.

[12] proposes *AutoScale* as a dynamic capacity management scheme that performs well in response to fluctuations in request rate, request size, and server efficiency, for example when there are internal service disruptions.

The scheme turns servers off conservatively, again imposing a wait time before turning off an idle on server. It routes incoming tasks primarily to a small subset of servers so that the other servers will time out and be shut down, thus preventing too many idle on servers. *AutoScale* is shown to maintain a good amount of excess capacity to handle spikes in incoming traffic and be robust to the above-listed fluctuations. *AutoScale* has a low 95^{th} percentile of response time, average power usage and average number of servers in use. [13] introduces the SOFTScale scheme which performs well during load spikes in multi-tier data centers without having to keep excess servers idle on. In a multi-tier data center, the data tier is always left on, while the servers in the application tier are sometimes turned off. SOFTScale takes advantage of this by using the data tier servers to do some application work when there is a load spike. SOFTScale is shown to be able to handle a significantly larger range of load jumps than if the scheme were not in place. During a load spike, SOFTScale is useful specifically when waiting for idle off servers to turn on, and can be coupled with other dynamic capacity management schemes.

With cloud networks centralized queues are not maintained and a parallel queue server system in which incoming tasks are sent to one of the queues immediately upon arrival at a central dispatcher is in place instead. Recently, research has been conducted on queuing algorithms for parallel queue server systems. The proposed schemes include the JIQ scheme and the TABS scheme.

The JIQ scheme is proposed in [14]. Through keeping track of an idle server queue, the central dispatcher has access to information on which servers are idle. This requires only constant communication overhead, unlike schemes like Join-the-Shortest-Queue (JSQ) which have large communication overhead in the parallel queue system setting. [15] proves that under Markovian assumptions and for a sub-critical load, the system process under JIQ is stable. In addition, as $N \to \infty$, in the steady state the expected wait time of a task vanishes in the limit. Using fluid limits, it is shown in [16], that the JIQ scheme is optimal at the diffusion level since it has the same diffusion-limit behavior as the JSQ scheme. However, the JIQ scheme does not provide a solution for inefficient server utilization rates and high energy consumption since all servers are left on at all times. This leaves room for a scheme that achieves efficient server utilization as well as system stability for finite N.

Proposed in [8], the TABS scheme is a token-based, auto-scaling, load-

balancing scheme that, like the JIQ scheme, has constant communication overhead. Similar to the M/M/N/setup/delayedoff scheme however, it turns servers that have been idle on for an exponentially distributed standby period off in order to achieve more efficient server utilization and lower energy consumption. Under the assumption of fixed size finite buffers, with suitable parameter values the fluid limit results coincide exactly with that of the JIQ scheme, whose fluid limit optimality is proven in [15, 17]. The fluid limit optimality of the TABS scheme is thus shown. As $N \to \infty$, the expected wait time and expected energy consumption vanish in the limit. The result of asymptotic stability for the TABS scheme requires the assumption of a fixed size finite buffer. [9] investigates the stability of the TABS scheme with infinite buffers instead. It is proved that for any fixed subcritical load, the system is stable for large enough N and, using this result, convergence of steady-state distributions to the optimal one, as $N \to \infty$ is established. It is noted that the result is only applicable to large N, the example of an unstable system with $N = 2$ is detailed on page 11.

All the previous work on distributed parallel queueing systems considers an asymptotic regime, where the number of servers tends to infinity. For many schemes, such as the JIQ scheme, this provides a good approximation for the finite system. However, as observed before, for the TABS scheme the asymptotic behavior may not necessarily reflect the finite-N scenario [9]. In fact, even such a fundamentally important property like stability may not hold for the TABS scheme. This necessitates more research into a scheme that achieves optimal service elasticity and stability in the finite-N case.

3 Key contributions

Motivated by the above considerations, we investigate stability in finite server systems. We propose the Stable Energy-Conserving, Thresholding (SECT) scheme for finite server systems that is stable in cases where the TABS scheme is not and provides greater robustness. It achieves better service elasticity in the finite number of servers case. Similar to TABS, the proposed SECT scheme is a token-based feedback protocol with constant communication overhead. The key difference is that the scheme keeps track of the level of congestion in the system and behaves similarly to the TABS scheme only when the congestion level is low.

Under the SECT scheme, the dispatcher keeps track of which servers are idle on, off, in set-up mode, busy but with queue length less than the threshold k, or busy and with queue length greater than or equal to k. When a server becomes idle, busy with a queue length under k, busy with a queue length at least k, off, or in set-up mode, it sends the relevant token corresponding to its current status to the central dispatcher to indicate its status. If a server is idle on for an $\text{Exp}(\mu)$ amount of time and all servers have indicated to the dispatcher that they are idle on, off, or busy with queue length under k, that server will be turned off. If a server is idle on for an $\text{Exp}(\mu)$ amount of time and there is at least one server that has indicated that it is busy with a queue length of at least k, the idle server will not be turned off and will remain idle on. When a task arrives and there is at least one switched on server that is idle, the dispatcher will send the task to one of the idle on servers at random. If all servers that are on are busy, the dispatcher will send the task to one of the busy servers at random, and will initiate the set-up procedure of one of the turned off servers, if there are any, at random. The time taken for the set-up procedure is exponentially distributed with rate ν, after which the server becomes idle on and sends the dispatcher the token corresponding to the idle on status.

Note that any incoming task is sent to an idle on or a busy server, and never to a server that is off. In addition, at most two tokens are sent per task, maintaining constant communication overhead.

The SECT scheme combines the beneficial aspects of the TABS and JIQ schemes. Like TABS, it allows for higher service utilization rates but at the same time, it provides the stability of the JIQ scheme. When the level of congestion in the system is high, the SECT scheme behaves like the JIQ scheme and when it is low it behaves similarly to the TABS scheme. Under the SECT scheme we observe that for a system with high congestion and a small number of servers, the longest queue lengths decrease at a rate similar to the JIQ scheme, while the under the TABS scheme they remain high. At the same time, the SECT scheme performs slightly better than the TABS scheme in terms of mean energy consumption and mean wait time. For this reason, using the SECT scheme is preferable to the TABS scheme when the number of servers in the system is small.

4 Details of Related Algorithms

Other parallel queue algorithms have been discussed in the literature review. Two schemes that are of particular interest are the popular Join-the-Idle-Queue (JIQ) scheme and the Token-based Auto Balance Scaling (TABS) Scheme.

JIQ Scheme:
In the JIQ scheme, all servers are kept on, even if they are idle for long periods of time. Under the JIQ scheme, idle servers send the central dispatcher a token. When a task arrives at the dispatcher, if there are any tokens, the dispatcher sends the task to one of the corresponding idle servers at random. If there are no tokens, this indicates that all of the servers are busy and the dispatcher sends the task to one of the busy servers at random.

TABS Scheme:
The TABS scheme is a token-based scheme in which idle on servers send the dispatcher tokens indicating their availability. They wait an Exp(μ) time, the standby time, before they turn off and send a new token to the dispatcher announcing their new switched off state. When a task arrives, the dispatcher sends it to a random idle on server which becomes busy. The server sends a different token to the dispatcher indicating its busy status. In the event that all servers that are on are busy, the dispatcher will send the task to random busy server. The dispatcher will also select a random server that is turned off, if there are any, and begin the set-up procedure for that server. The set-up procedure has an Exp(ν) distribution. That server will send a token indicating that it is in set-up mode to the dispatcher which will be replaced by the token corresponding to idle on once the set-up procedure is complete.

We will now investigate the behavior of the TABS scheme in the case where there are a small number of servers. Consider the case in which there are two servers, A and B, with parallel queues and an arrival rate to each server of $\frac{1}{2} < \lambda < 1$. Let A start with an initial large number of tasks in its queue and B start with a small number of tasks. Every time all of the tasks in B's queue are serviced, B will be in an idle on state while A remains busy. The event that B is idle on for an Exp(μ) amount od time and turns off before the next task arrives occurs with positive probability. Thus the next task that arrives will be sent to A with an arrival rate to server A of $2\lambda > 1$. B will begin the setup process, but as this has an exponential distribution with rate ν, there is again a positive probability of more tasks arriving and being

added to A's queue. The expected number of tasks sent to A is at least $\frac{2\lambda}{\nu}$ and the expected number of departures is $\frac{1}{\nu} < \frac{2\lambda}{\nu}$. Thus, the expected value of the increase in A's queue length while B is in setup mode is $\frac{2\lambda-1}{\nu}$. Furthermore, once B has turned on and is busy, it is likely that its queue will empty out again and repeat the above process since now the arrival rate to each server is again $\frac{1}{2} < \lambda < 1$ as in the initial scenario. This leads to the above sequence of events repeating and A's queue length increasing, indicating instability. A similar issue arises for other finite values of N.

5 Details of Model and Algorithm

Model Description. Consider a system that consists of a central dispatcher and N identical servers with parallel queues. Tasks arrive at the central dispatcher as a Poisson process with rate $\lambda_N(t) = N\lambda(t)$, for time $t \geq 0$, where $\lambda(\cdot)$ is a bounded positive real-valued function, bounded away from zero. In this paper we assume that the arrival rate is constant and does not change with time i.e. $\lambda(t) \equiv \lambda$, where λ is a constant. A task arriving at the central dispatcher cannot be queued there and is immediately sent to one of the servers' queues. Servicing the task is exponentially distributed with rate 1 after which the server starts servicing the next task in the queue, or becomes idle on if the queue is empty. A server that is turned off takes setup time $\text{Exp}(\nu)$ to turn on.

Let $X_i(t)$ represent the length of the i^{th} queue at time t, $1 \leq i \leq N$, and let k be the threshold queue length. Let $Q_m^N(t)$ be the number of servers with queue length greater than or equal to m at time t including any task that is being serviced at time t. We note that $Q_1^N(t)$ is the number of busy servers at time t.

In order to analyze the stability of the SECT scheme, we look at the evolution of the maximum queue length, $\max_{1\leq i\leq N}(X_i(t)),\ \ 0 \leq t \leq T$. In particular, we will call a scheme stable if under the scheme, the following holds in the system:

Start from an initial state where $\exists i : X_i(0) \geq k$ meaning that at least one server has a queue length greater than or equal to k. Let $\eta = \inf\{t \geq 0 : X_i(t) < k \quad \forall i\}$. η is the first time at which all queue lengths to decrease to below k. If $\mathbb{E}[\eta] < \infty$, then the scheme is stable.

We will now introduce the SECT scheme which provides the flexibility of the JIQ scheme as well as the efficient server utilization of the TABS scheme.

Algorithm Specification. SECT:

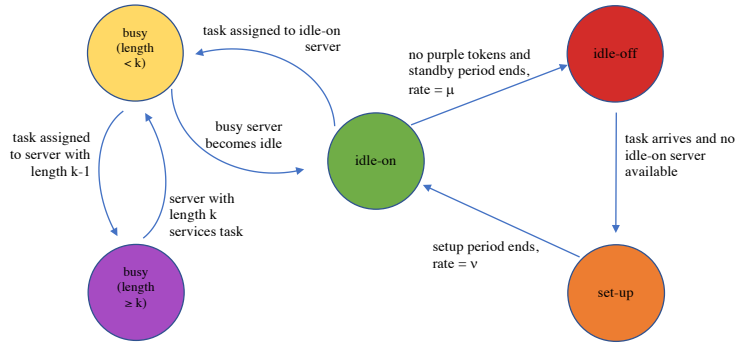

Figure 1: Illustration of server on-off decision rules in the SECT scheme, along with token colors

- There is a constant threshold queue length k.
- When a server becomes idle on, it sends a 'green' token to the dispatcher. After waiting the Exp(μ) standby period, if there are no 'purple' tokens at the dispatcher (explained below), the server turns itself off and sends a 'red' token to the dispatcher. That server's initial 'green' token is destroyed. If there are 'purple' tokens at the dispatcher, the server remains on and idle and the 'green' token is retained.
- When the dispatcher receives an incoming task, if there are any 'green' tokens, it selects a 'green' token uniformly at random and sends the task to that server. The server becomes busy and if it has queue length under k (for $k > 1$), it sends a 'yellow' token to the dispatcher. The respective 'green' token is destroyed. If there are no 'green' tokens (all the servers are busy or off), the dispatcher selects a token uniformly at random from the pool of 'yellow' and 'purple' tokens, and sends the task to the corresponding server. The server remains busy and if it has queue length under k, it sends a 'yellow' token to the dispatcher, if it has a queue length greater than or equal to k, it sends a 'purple' token to the dispatcher.

- If a task is sent to a busy server i.e. a 'yellow' or 'purple' token is selected, if there are any 'red' tokens at the dispatcher, the dispatcher picks a 'red' token uniformly at random and initiates the start-up procedure of that sever, which takes an Exp(ν) amount of time. The 'red' token for that server is replaced by an 'orange' token. When the start-up procedure is complete, a 'green' token is sent to the dispatcher and the now idle on server begins waiting the duration of the standby period.
- If a busy server with a 'yellow' token finishes servicing a task and becomes idle, it sends the dispatcher a 'green' token. If a busy server with a 'purple' token finishes servicing a task and now has queue length $= k-1$, it replaces its 'purple' token at the dispatcher with a 'yellow' token.

6 Theoretical Results

Outline: The proof of stability as defined in Section 5 is structured in three parts as follows. We will first prove that starting from a state of instability where at least one queue length is greater than or equal to the threshold k, given that at least one queue length remains at least k, in expected finite time T' the number of idle off servers will go to 0. We then prove that starting from time T', the number of servers in setup mode will go to 0, given that at least one queue length remains greater than or equal to k. The final part of the proof shows that starting from time T'', a state with all servers on and at least one queue length greater than or equal to k, the system behaves as it would under the JIQ scheme and the number of servers with queue lengths greater than or equal to k will go to 0 in expected finite time showing that the SECT scheme is stable.

Lemma 6.1. *Suppose the system starts in a state where there is at least one queue that is greater than or equal to the threshold length k i.e.* $Q_k^N(0) > 0$. *Let* τ *be the first time that the system reaches a state in which either all queue lengths are below the threshold, or all servers are either busy or idle on.*

$$\tau := \inf\{t \geq 0 : \Delta_1^N(t) = 0 \text{ and } \Delta_0^N(t) = 0 \text{ or } Q_k^N(t) = 0\}$$

Then $\mathbb{E}[\tau] < \infty$.

Claim 6.1.1. *Start from a state at time* 0, *where there are* N *servers total,* $Q_1^N(0)$ *busy servers,* $U^N(0)$ *idle on servers,* $\Delta_0^N(0)$ *idle off servers*

and $\Delta_1^N(0)$ servers in setup mode (where $Q_1^N(0), U^N(0), \Delta_0^N(0), \Delta_1^N(0) \geq 0, Q_1^N(0) + U^N(0) + \Delta_0^N(0) + \Delta_1^N(0) = N$); and at least one queue is at least the threshold length k. Then either all queue lengths will go below k, or the number of off servers will decrease to 0 in finite expected time.

$$T' := \inf\{t \geq 0 : \Delta_0^N(t) = 0 \text{ or } Q_k^N(t) = 0\}$$

We want to show that $\mathbb{E}[T'] < \infty$.

Proof. We will begin by upper bounding $\mathbb{P}(T' \geq t)$. First we note that in the interval $[0, t]$, there is at least one queue length greater than or equal to the threshold k, otherwise $T' < t$. Thus, under the SECT scheme no idle on servers will be switched off regardless of how long they have been idle on for in the interval $[0, t]$.

$$\begin{aligned} S_{[t_i,t_j]} &:= \text{number of tasks arriving to the system in interval } [t_i, t_j] \\ S^B_{[t_i,t_j]} &:= \text{number of tasks arriving to busy servers in interval } [t_i, t_j] \\ C_i &:= \text{service time for task } i \end{aligned}$$

Consider the interval $[0, 1]$. Define $\mathcal{A}$ to be the event that in the interval, fewer than $\Delta_0^N(0)$ arriving tasks are sent to busy servers. Define $\mathcal{B}$ to be the event that in the interval, at least one queue length is greater than or equal to k. In terms of the notation defined above:

$$\begin{aligned} \mathcal{A} &:= \{S^B_{[0,1]} < \Delta_0^N(0)\} \\ \mathcal{B} &:= \{Q_k^N(t) > 0 \quad \forall t \in [0, 1]\} \end{aligned}$$

The event $T' > 1$ is the event that in the interval $[0, 1]$, there is at least one server in idle off mode, and at least one queue length is greater than or equal to k.

Thus $(T' > 1) = (\mathcal{A} \cap \mathcal{B})$, and $(T' > 1) \implies \mathcal{A}$. Thus, uniformly in $Q_1^N(0), U^N(0), \Delta_0^N(0), \Delta_1^N(0)$:

$$\begin{aligned}\mathbb{P}(T' > 1) &\leq \mathbb{P}(\mathcal{A}) \\ &= 1 - \mathbb{P}(\mathcal{A}^{\mathsf{c}})\end{aligned}$$

Define $\mathcal{C}$ to be the event that N tasks arrive in the interval $[0, 1]$ i.e.

$$\mathcal{C} := \{S_{[0,1]} = N\}$$

Let the N tasks arriving in [0,1] if event $\mathcal{C}$ occurs be $j_1, ..., j_N$.

$$J_d := \text{ number of the } j_1, ..., j_N \text{ tasks that depart in } [0, 1]$$

We define $\mathcal{D}$ to be the event that none of the N new tasks depart in the interval $[0, 1]$. Let $\mathcal{D}'$ be the event that each task arriving in $[0, 1]$ takes more than 1 second to be serviced. Using the notation defined above:

$$\begin{aligned}\mathcal{D} &:= \{J_d = 0\} \\ \mathcal{D}' &:= \{C_{j_i} > 1 \quad i = 1, ..., N\}\end{aligned}$$

We know that $C \perp\!\!\!\perp D'$, and $(\mathcal{C} \cap \mathcal{D}') \implies (\mathcal{C} \cap \mathcal{D}) \implies \mathcal{A}^{\mathsf{c}}$.
Thus, we know,

$$\begin{aligned}\mathbb{P}(\mathcal{A}^{\mathsf{c}}) &\geq \mathbb{P}(\mathcal{C} \cap \mathcal{D}) \\ &\geq \mathbb{P}(\mathcal{C} \cap \mathcal{D}') \\ &= P(\mathcal{C})\mathbb{P}(\mathcal{D}') \\ &\geq \frac{e^{-\lambda N}(\lambda N)^N}{N!} \cdot (e^{-N}).\end{aligned}$$

Therefore,

$$\begin{aligned}\mathbb{P}(T' > 1) &\leq \mathbb{P}(\mathcal{A}) \\ &\leq 1 - \frac{e^{-N(\lambda+1)}(\lambda N)^N}{N!}\end{aligned}$$

Consider t such independent $[0, 1]$ intervals: $[0, 1], [1, 2], ..., [t - 1, t]$, where $\mathcal{E}$ is defined as the event that in every one of the t intervals, the number

of idle off servers > 0 and at least one queue length is greater than k. Let $\mathcal{F}(i), i = 0, 1, ..., t-1$ be the event that fewer than $\Delta_0^N(i)$ arriving tasks are sent to busy servers in the interval $[i, i+1]$. That is:

$$\mathcal{E} := \{\Delta_0^N(s) > 0, Q_k^N(s) > 0 \quad \forall s \in [i, i+1], i = 0, 1, ..., t-1\}$$
$$\mathcal{F}(i) := \{S^B_{[i,i+1]} < \Delta_0^N(i)\} \quad i = 0, 1, ...t-1$$

From above we know that

$$F(i) \leq 1 - \frac{e^{-N(\lambda+1)}(\lambda N)^N}{N!}$$

$$\mathcal{F}(0) \perp\!\!\!\perp \mathcal{F}(1) \perp\!\!\!\perp ... \perp\!\!\!\perp \mathcal{F}(t-1)$$

.
Since $(T' > t) \implies \mathcal{E} \implies (\mathcal{F}(0) \cap \mathcal{F}(1) \cap ... \cap \mathcal{F}(t-1))$,

$$\begin{aligned}
\mathbb{P}(T' > t) &\leq \mathbb{P}(\mathcal{E}) \\
&\leq \mathbb{P}(\mathcal{F}(0) \cap \mathcal{F}(1) \cap ... \cap \mathcal{F}(t-1)) \\
&\leq (1 - \frac{e^{-N(\lambda+1)}(\lambda N)^N}{N!})^t
\end{aligned}$$

From this we see that $\mathbb{P}(T' > t)$ is a non-increasing function.

$$\begin{aligned}
\mathbb{E}[T'] &= \int_0^\infty \mathbb{P}(T' \geq t)dt, t \geq 0 \\
&\leq \sum_{s=1}^{\infty} \mathbb{P}(T' \geq s) \cdot 1 \\
&\leq \sum_{s=1}^{\infty} (1 - \frac{e^{-N(\lambda+1)}(\lambda N)^N}{N!})^s \\
&< \infty \text{ due to the non-increasing nature of the function}
\end{aligned}$$

□

Claim 6.1.2. *Start from a state at time 0 with* $0 \leq \Delta_1^N(0) \leq N$ *and* $Q_k(0) > 0$.

$$\tau := \inf\{t \geq 0 : \Delta_1^N(t) = 0 \text{ and } \Delta_0^N(t) = 0 \text{ or } Q_k(t) = 0\}$$

Then $\mathbb{E}[\tau] < \infty$.

Proof. From *Claim 6.1.1* we know that $\mathbb{E}[T'] < \infty$, where

$$T' := \inf\{t \geq 0 : \Delta_0^N(t) = 0 \text{ or } Q_k(t) = 0\}$$

If it is the case that $Q_k(T') = 0$, then *Claim 6.1.2* automatically holds. In the other case, $\Delta_0^N(T') = 0$ and $Q_k(T') > 0$. We will now consider the second case and define $\bar{T} = T'$ in this case.

Consider the Markov Chain where each state represents the number of servers in setup mode. The Markov Chain starts in state $\Delta_1^N(\bar{T})$. Starting from this state, either the queues all go below threshold length k in finite time in which case *Claim 6.1.2* holds, or there is at least one server with queue length at least k. In the latter case, no servers will be turned off and since $\Delta_0^N(\bar{T}) = 0$, we know that the number of servers in setup mode is non-increasing.

$$T'' := \inf\{t \geq \bar{T} : \Delta_1^N(t) = 0 \mid Q_k^N(s) > 0 \quad \forall s \in (\bar{T}, t)\}$$

The Markov Chain is shown below.

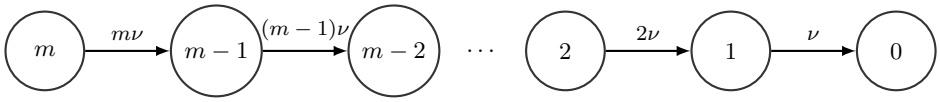

Figure 2: Markov Chain representation of the number of servers in setup mode, where the initial state $0 \leq m \leq N$

From this we can find $\mathbb{E}[T'']$. We know that the expected time spent in state $i = \frac{1}{i\nu}$ since the service times follow an exponential ν distribution. Thus, starting from state $\Delta_1^N(\bar{T})$,

$$\mathbb{E}[T'' \mid \Delta_1^N(\bar{T})] = \bar{T} + \sum_{i=1}^{\Delta_1^N(0)} \frac{1}{i\nu} < \infty$$

.

$$\sup_{Q_1^N(0),U^N(0),\Delta_0^N(0),\Delta_1^N(0)} \mathbb{E}[T'' \mid \Delta_1^N((\bar{T}))] = \bar{T} + \sum_{i=1}^{N} \frac{1}{i\nu} < \infty \quad \text{since } N < \infty$$

Since we know $T'' \geq \tau$,

$$\mathbb{E}[T''] < \infty \implies \mathbb{E}[\tau] < \infty$$

□

Thus the expected time taken for all idle off and servers in setup mode to turn on, or all queue lengths to go below k is finite which is what we wanted to show in *Lemma 6.1.*

Assumption 6.2. *Suppose the system starts in an initial state in which* $\Delta_0^N(0) = 0$, $\Delta_1^N(0) = 0$, $Q_k^N(0) > 0$ *and* $\lambda < 1$.

$$\eta := \inf\{t \geq 0 : Q_k^N(t) = 0\}$$

Then $\mathbb{E}[\eta] < \infty$.

Proof heuristics. Starting from a state with $\Delta_0^N(0) = \Delta_1^N(0) = 0$, until time η all servers are switched on and no servers are permitted to turn off. Thus, the system under the SECT scheme behaves exactly as it would under the well-known JIQ scheme which is known to be stable. Hence $\mathbb{E}[\eta] < \infty$.

Let $\mathcal{K}$ be the finite collection of states in which all servers' queue lengths are less than k. We know that within $\mathcal{K}$, the system is irreducible and aperiodic which means that it is also positive recurrent, implying ergodicity. From *Lemma 6.1* and *Assumption 6.2*, we know that for any initial condition $Q_1^N(0), U^N(0), \Delta_0^N(0)$, $\Delta_1^N(0)$, $\mathbb{E}[\eta] < \infty$ meaning that the expected time taken for the system to enter set $\mathcal{K}$ is finite. Since the system is well-behaved in $\mathcal{K}$ and returns to $\mathcal{K}$ in expected finite time, we have proved the stability of the SECT scheme.

7 Simulation Experiments

In this section, the simulation results are presented. We discuss how the maximum queue length, $\max_{1\leq i\leq N}(X_i(t)), 0 \leq t \leq T$, evolves with time, as well as the number of waiting tasks in the system, $\sum_{i=1}^{N} X_i(t)$. T is the maximum time the simulation is run for. We also compare the performance of the SECT scheme to that of the TABS scheme using the expected energy consumption and expected wait time metrics. The threshold used for the SECT scheme is $k = 2$.

Maximum queue length decreases when starting from a state with a large number of tasks in the system. The maximum queue lengths, $\max_{1\leq i\leq N}(X_i(t)), 0 \leq t \leq T$, for the SECT scheme, TABS scheme and JIQ scheme are illustrated in Figure 3. The system at time $t = 0$ has $X_i(0) = 100, i = 1, ..., 5$ and $X_i(0) = 1, i = 6, ..., 10$ for all three schemes. The simulation is run for a maximum time of 1000, the standby period rate $\mu = 0.3$, and the mean setup time $\nu^{-1} = 100$. We observe that the maximum queue length under the SECT scheme decreases from 100 to values near 0 in a time similar to that of the JIQ scheme. On the other hand, the maximum queue length for the TABS scheme remains around the initial level of 100. Figure 4 plots the total number of waiting tasks in the system for the three schemes. We see that this metric again decreases to 0 in a similar manner for the SECT and JIQ schemes, while the decrease under the TABS scheme takes a longer time. These results are in accordance with the theoretical results presented in the previous section. At time $t = 0$, the system is congested as half of the servers have high queue lengths well above k meaning that under the SECT scheme, no servers are allowed to be turned off and correspondingly, all servers remain on. Thus the SECT scheme behaves like the JIQ scheme. Under the TABS scheme the servers that have $X_i(0) = 1$ initially may finish servicing the small number of tasks in their queues and remain idle on for the exponentially distributed standby period and turn off. This means that incoming tasks will be sent to busy servers which already have long queues, thus delaying the decrease of the maximum queue length. This also leads to a higher number of total waiting tasks since the tasks sent to the long queues will have to remain in the system for a longer amount of time before being serviced. As explained above, this issue is avoided under the SECT scheme.

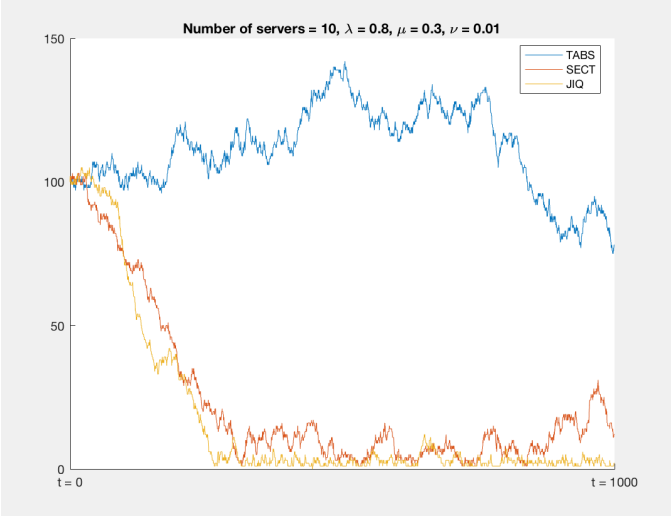

Figure 3: Comparison of the evolution of the maximum queue lengths for TABS, SECT and JIQ, with $N = 10$ servers, arrival rate $\lambda = 0.8$, standby period rate $\mu = 0.3$, setup period rate $\nu = 0.01$

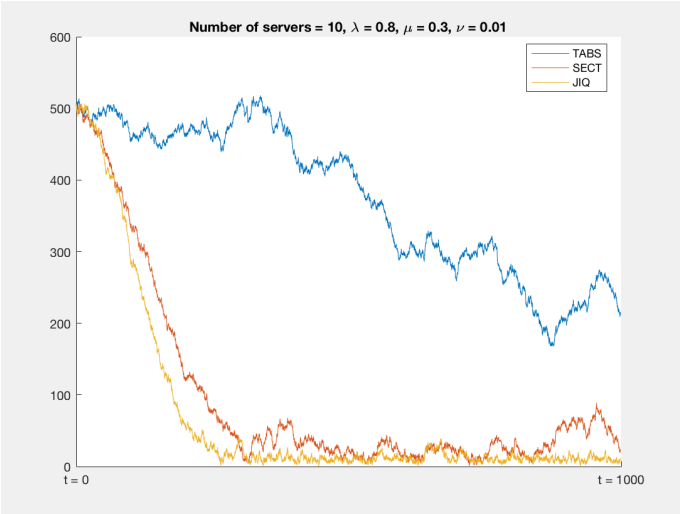

Figure 4: Comparison of the number of waiting tasks in the system for TABS, SECT and JIQ, with $N = 10$ servers, arrival rate $\lambda = 0.8$, standby period rate $\mu = 0.3$, setup period rate $\nu = 0.01$

Figure 5 again plots the maximum queue lengths for the SECT, TABS and JIQ schemes. Here we see small 'bumps' in the graph for the SECT scheme where once the maximum queue length decreases to near 0 it then increases again before decreasing back to near 0. This behavior can be explained by the fact that when there is a large number of tasks in the system, as is the case when $t = 0$ and at least one queue has length at least k, the SECT scheme behaves like the JIQ scheme and the maximum queue length decreases. Once the number of tasks in the system has decreased so that all queues are below the threshold i.e. the maximum queue length is less than k, the scheme behaves similarly to the TABS scheme which can lead to the maximum queue length increasing. However, once the maximum queue length has increased to at least k, the SECT scheme again begins acting like the JIQ scheme and the maximum queue length will again decrease as we proved in the previous section.

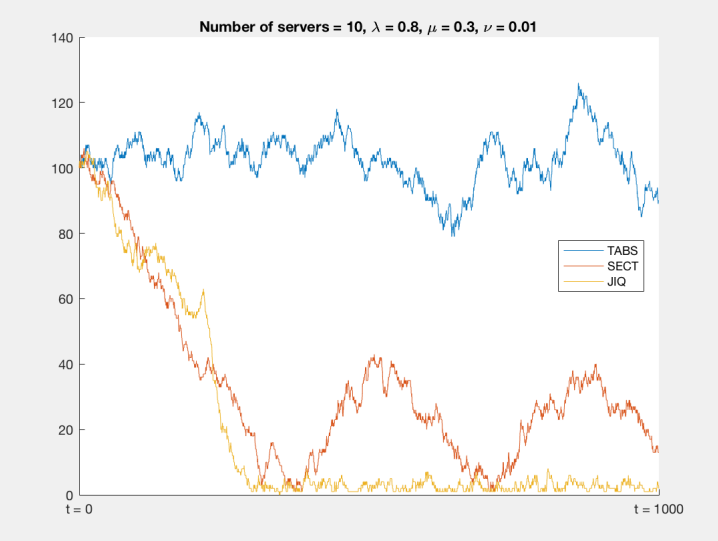

Figure 5: Evolution of the maximum queue lengths where there is a spike and subsequent decrease in the maximum queue length for the SECT scheme, with $N = 10$ servers, arrival rate $\lambda = 0.8$, standby period rate $\mu = 0.3$, setup period rate $\nu = 0.01$

To examine this behavior further, Figure 6 plots the maximum queue lengths for the SECT and TABS schemes when $X_i(0) = 1 \quad \forall 1 \leq i \leq 10$, meaning that $Q_k^N(0) = 0$. Thus the SECT scheme initially behaves like the TABS

scheme and we can see from the figure that for the reasons explained above, under the SECT scheme the maximum queue length initially increases. However, once $Q_k^N(t) > 0$, servers are prevented from turning off and the maximum queue length begins decreasing. For the TABS scheme, similar to the SECT scheme, the maximum queue length initially increases. However, it does not decrease again like it does under the SECT scheme since servers are permitted to turn off.

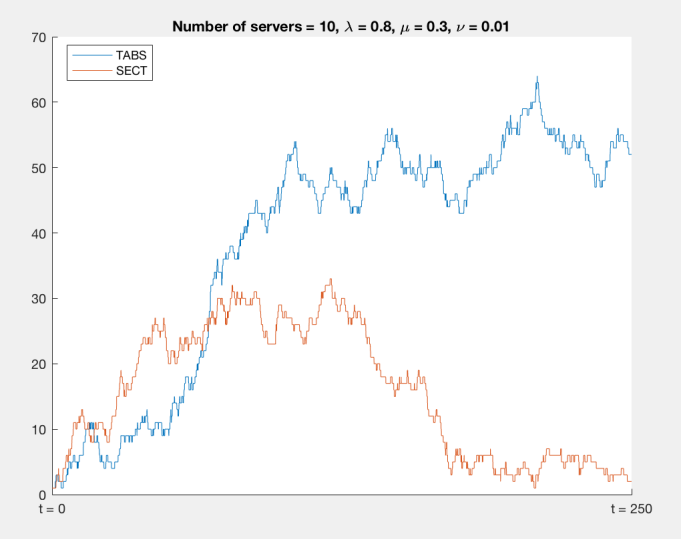

Figure 6: Comparison of the behavior of the SECT and TABS schemes when the maximum queue length begins to increase

Effect of λ on performance of the SECT scheme.
From Figure 7, we see that when the arrival rate of tasks into the system is high with $\lambda = 1$, the SECT scheme still performs in a similar manner to the JIQ scheme in terms of maximum queue length and number of waiting tasks in the system, which we use as a measure of stability. Under the TABS scheme, the maximum queue length as well as the number of waiting tasks in the system trend upwards. The mean fraction of idle off servers for the TABS scheme was 4.5755×10^{-5} and 0 for the SECT scheme. This implies that the SECT scheme behaved like the JIQ scheme over the course of the simulation when $\lambda = 1$ since the maximum queue length remained above k, which explains their similar performance. This simulation was again run with $X_i(0) = 100, i = 1, ..., 5$ and $X_i(0) = 1, i = 6, ..., 10$ for a time of 1000, and with $N = 10, \mu = 0.3, \nu = 0.01$ for each of the three schemes.

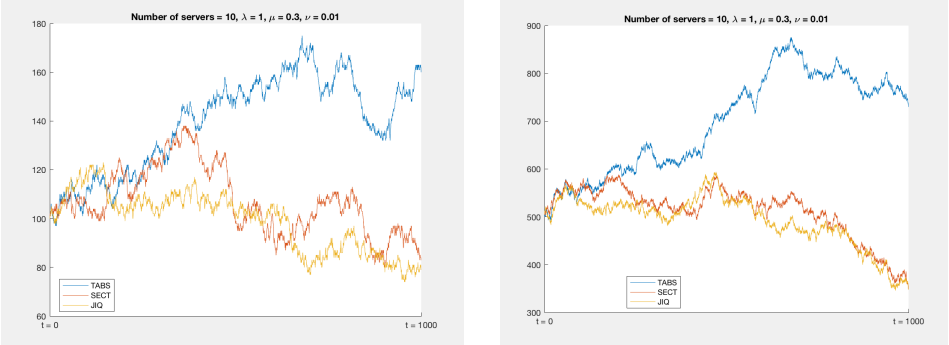

Figure 7: Comparison of maximum queue lengths and number of waiting tasks when $\lambda = 1$

Figure 8 plots the maximum queue length and number of waiting tasks in the system when when the arrival rate of tasks into the system is low with $\lambda = 0.6$ for the three schemes. We see that the SECT scheme still performs better than the TABS scheme in terms of stability, though the improvement is less significant than when λ is higher. Theoretically this is because the poor performance of the TABS scheme arose when servers were turned off after being left idle on for the standby period, but tasks kept arriving and had to be sent to servers whose queue lengths were already high. With a lower λ however, this is less of an issue since with a slower arrival rate, servers that have been turned off can turn back on before a large number of incoming tasks are sent to already busy servers. This simulation was again run with $X_i(0) = 100, i = 1, ..., 5$ and $X_i(0) = 1, i = 6, ..., 10$ for a time of 1000, and with $N = 10, \mu = 0.3, \nu = 0.01$ for each of the three schemes.

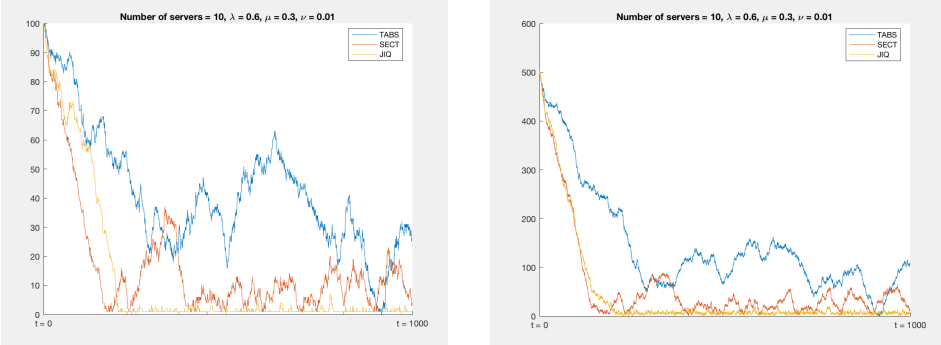

Figure 8: Comparison of maximum queue lengths and number of waiting tasks when $\lambda = 0.6$

Energy consumption and mean expected wait time under the SECT scheme is lower than under TABS. In order to quantify the energy usage, the following values of power consumption are used. A server that is on and busy or in setup mode consumes $P_{full} = 200W$, a server that is on and idle consumes $P_{idle} = 140W$, and a server that is off consumes no power. In Figure 9, average values of energy consumption over 2000 seconds have been plotted as a function of the mean standby period, μ^{-1}. We can observe that the average energy consumption under the SECT scheme is less than that under the TABS scheme for all values of μ^{-1}. In addition, the difference is more pronounced for smaller values of μ^{-1}. This observation is likely explained by the fact that under the SECT scheme, when there is high congestion in the system servers are prevented from turning off, which is not the case for TABS. This means that in times of high incoming traffic, these servers that have been turned off are likely to have to be turned on again relatively quickly under the TABS scheme which consumes power, P_{full}. By preventing the unnecessary turning off of these servers in the first place, the high energy cost of the setup procedure is avoided under the SECT scheme and these servers would consume the lower P_{idle} instead. This is especially true when μ^{-1} is small since idle servers only wait a short amount of time before turning off, which increases the number of unnecessary turn offs under the TABS scheme.

Expected wait time is calculated as $\mathbb{E}[W] = \lambda^{-1} \sum_{i=1}^{3} q_i$. In Figure 10, the average expected wait times over 2000 seconds have been plotted as a function of the mean standby period, μ^{-1}. We observe that the SECT scheme performs better than the TABS scheme in terms of expected wait time for

all values of μ^{-1}. This again is especially true when μ^{-1} is small. When there is high traffic in the system and $Q_k^N(t) > 0$, under the SECT scheme, no servers will be turned off at time t. As shown in Section 6, either all queues will go below the threshold level or all servers will be on in expected finite time. This means that in expected finite time, either all queues in the system decrease to relatively short lengths or all available resources are used to service tasks, both of which lead to lower wait times. However, this is not true of the TABS scheme under which even in high traffic scenarios, idle on servers will turn off after waiting the standby amount of time. This reduces the number of available servers for tasks to be routed to and thus longer wait times. This is especially true when μ^{-1} is small since idle on servers will be turned off relatively quickly under the TABS scheme.

Thus we observe that when N is small, the SECT scheme provides better user-perceived performance than the TABS scheme whilst improving energy efficiency.

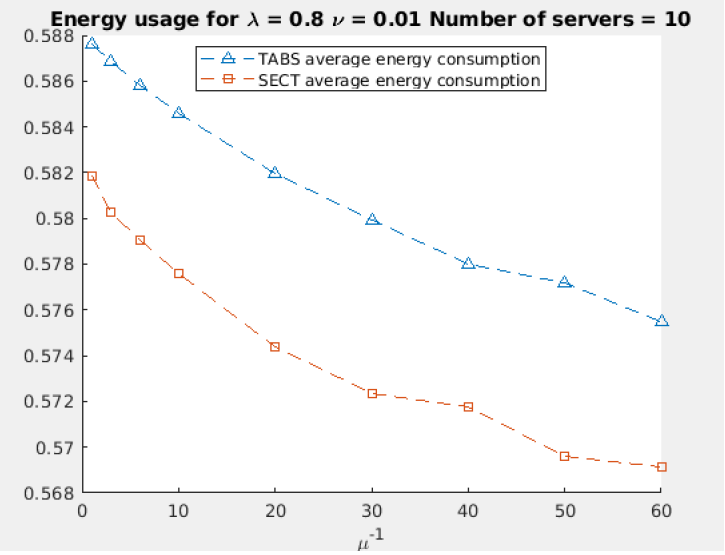

Figure 9: Energy usage for $N = 10$ servers, mean setup period $\nu^{-1} = 100$, arrival rate $\lambda = 0.8$, as a function of mean standby period μ^{-1}

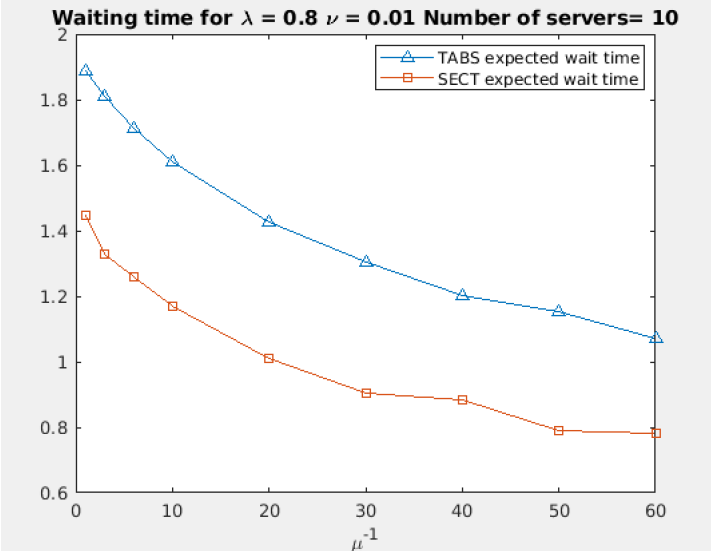

Figure 10: Expected wait time for $N = 10$ servers, mean setup period $\nu^{-1} = 100$, arrival rate $\lambda = 0.8$, as a function of mean standby period μ^{-1}

8 Conclusion and Open Questions

It is commonplace for data centers to face low server utilization rates, which result in high energy consumption and the related financial and environmental costs. This has resulted in centralized queue-driven auto-scaling schemes that improve server utilization rates whilst maintaining excellent user-facing performance. However, the rise of cloud networks necessitates auto-scaling, load-balancing schemes that work in systems with distributed parallel queue systems, rather than centralized queue-driven auto-scaling techniques. It is unclear that systems with a small number of servers will be stable under the token-based joint auto-scaling and load balancing TABS scheme currently proposed in literature. Motivated by these considerations, we proposed the SECT scheme which does not require global queue length information and achieves higher server utilization rates and constant communication overhead like the TABS scheme. At the same time however, we prove that the scheme achieves stability in parallel queue systems with a small number of servers. The proofs relied on lower bounding a CDF and using a Continuous-time Markov Chain representation to show that the system under the SECT scheme can be coupled to the system under the JIQ scheme.

The simulation experiments conducted corroborate the theoretical stability results. We observe that the maximum queue length under the SECT scheme behaves in a similar manner to that of the JIQ scheme for a range of arrival rates. At the same time, it performs better than the TABS scheme in terms of both energy consumption and mean expected wait time across a range of standby period rates.

It would be of interest for future research to investigate how the SECT scheme performs as $N \to \infty$ relative to the TABS scheme. I conjecture that the scheme will be stable as $N \to \infty$ and will perform better in terms of expected average wait time. The effect on expected energy consumption is less clear, but I believe that it will perform comparably to the TABS scheme depending on the threshold chosen.

Other areas of future research include the effect of $\lambda(t)$ that varies with t, as well as standby periods that are variable and responsive to the current state of the system. We could also investigate the SECT scheme with a slight modification in which incoming tasks are distributed amongst servers with queue lengths under the threshold if there are any, rather than amongst all busy servers. It would be interesting to determine whether it has a significant impact on the average expected wait time of tasks. In investigating the stability of the modified SECT scheme and the SECT scheme with a time-dependent $\lambda(t)$, proving that the system can be coupled to the system under the JIQ scheme would become more complex.

9 Bibliography

References

[1] Yevgeniy Sverdlik. Here's how much energy all us data centers consume. https://www.datacenterknowledge.com/archives/2016/06/27/heres-how-much-energy-all-us-data-centers-consume, June 2016.

[2] João Marques Lima. Data centres of the world will consume 1/5 of earth's power by 2025. https://data-economy.com/data-centres-world-will-consume-1-5-earths-power-2025/, December 2017.

[3] Energy hogs: Can world's huge data centers be made more efficient? https://e360.yale.edu/features/energy-hogs-can-huge-data-centers-be-made-more-efficient, April 2018.

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

[5] U.S Chamber of Commerce Technology Engagement Center. Data centers: Jobs and opportunities in communities nationwide, 2017.

[6] Green Grid. Unused servers survey results analysis. http://www.thegreengrid.org/en/Global/Content/white-papers/UnusedServersSurveyResultsAnalysis, 2010.

[7] Anshul Gandhi, Mor Harchol-Balter, and Michael A. Kozuch. Are sleep states effective in data centers? In *Proceedings of the 2012 International Green Computing Conference (IGCC)*, IGCC '12, pages 1–10, Washington, DC, USA, 2012. IEEE Computer Society.

[8] Debankur Mukherjee, Souvik Dhara, Sem C. Borst, and Johan S.H. van Leeuwaarden. Optimal service elasticity in large-scale distributed systems. *SIGMETRICS Perform. Eval. Rev.*, 45(1):3–3, June 2017.

[9] Debankur Mukherjee and Alexander Stolyar. Join-idle-queue with service elasticity: Large-scale asymptotics of a non-monotone system. 03 2018.

[10] Anshul Gandhi, Mor Harchol-Balter, and Ivo Adan. Server farms with setup costs. *Perform. Eval.*, 67(11):1123–1138, November 2010.

[11] Anshul Gandhi, Sherwin Doroudi, Mor Harchol-Balter, and Alan Scheller-Wolf. Exact analysis of the m/m/k/setup class of markov chains via recursive renewal reward. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, pages 153–166, New York, NY, USA, 2013. ACM.

[12] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A. Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Trans. Comput. Syst.*, 30(4):14:1–14:26, November 2012.

[13] Anshul Gandhi, Timothy Zhu, Mor Harchol-Balter, and Michael A Kozuch. Softscale: Stealing opportunistically for transient scaling. pages 142–163, 12 2012.

[14] Yi Lu, Qiaomin Xie, Gabriel Kliot, Alan Geller, James Larus, and Albert Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Perform. Eval.*, 68:1056–1071, 11 2011.

[15] Alexander L. Stolyar. Pull-based load distribution among heterogeneous parallel servers: The case of multiple routers. *Queueing Syst. Theory Appl.*, 85(1-2):31–65, February 2017.

[16] D. Mukherjee, S. C. Borst, J. S. H. van Leeuwaarden, and P. A. Whiting. Universality of Load Balancing Schemes on Diffusion Scale. *arXiv e-prints*, page arXiv:1510.02657, Oct 2015.

[17] Alexander L. Stolyar. Pull-based load distribution in large-scale heterogeneous service systems. *Queueing Syst. Theory Appl.*, 80(4):341–361, August 2015.