Power Minimization for Data Center with Guaranteed QoS

Shuo Liu*, Soamar Homsi*, Ming Fan†, Shaolei Ren‡, Gang Quan*, Shangping Ren§
*Department of Electrical and Computer Engineering, Florida International University, Miami, FL, 33174
†Broadcom Corporation, 3151 Zanker Road, San Jose, CA, 95134
‡School of Computing and Information Sciences, Florida International University, Miami, FL, 33199
§Department of Computer Science, Illinois Institute of Technology, Chicago, IL, 60616
Emails: {sliu005, shoms001, gang.quan}@fiu.edu, mingfan@broadcom.com, sren@cs.fiu.edu, ren@iit.edu

***Abstract*—Data centers have been widely employed to offer reliable and agile on-demand web services. However, the dramatic increase of the operational cost, largely due to the power consumptions, has posed a significant challenge to the service providers as services expand in both scale and scope. In this paper, we study the problem of how to improve resource utilization and minimize power consumption in a data center with guaranteed quality-of-service (QoS). Different from a common approach that separates requests with different QoS levels on different servers, we devise an approach to pack requests of the same service — even with different QoS requirements — into the same server to improve resource usage. We also develop a novel method to improve the system utilization without compromising the QoS levels by removing potential failure requests. Experimental results show superiority of our approach over other widely applied approaches.**

I. Introduction

Service providers have been striving to enrich their service product lines since the outset to greatly facilitate people's lives. As web services expand in both scale and scope, resource management becomes more and more critical but challenging. Low resource utilization and high power consumption in data centers (where web services are hosted) are well-known issues. For example, there are more than 500K servers in Google's data centers cost more than $38M worth of electricity each year [1]. However, even for the web giant "Google", who has already investigated a significant amount of effort on making its data centers greener, it only has an average CPU utilization of around 40% [2][3]. On the other hand, while saving power consumption is important, service providers must also provide services that can satisfy user's QoS requirements, such as response time and/or deadline miss ratios. This is particularly critical for those highly interactive services such as on-line gaming and on-line streaming multimedia (e.g. Pocket Gems and Sony Music on Google Cloud Platform [4][5]), where the user experience is closely related to the response times of the services. While these services are soft real-time in nature, it is imperative for service providers to ensure that adequate requests can be served successfully in time in order to fulfill QoS requirements.

There are extensive studies conducted to improve service provider's power efficiencies (e.g. [6][7][8][9][10]). Server consolidation, for example, has been a common approach to achieve high power/energy efficiency for data centers. The virtualization technology has enabled multiple virtual machine instances be executed on the same physical server. Since a server's power consumption is not exactly proportional to its utilization, and server may consume a significant amount of power even at a very low level of utilization, consolidation helps to increase the utilization of a server and minimize the number of activated physical servers needed. For example, in [6], Verma et al. present consolidation decisions that are made based on the correlations among different workloads. In [7], Mastroianni et al. devise two probability functions to model the effects of virtual machine allocations and migrations. By statistically analyzing the need of a virtual machine allocation or migration, they minimize the number of powered-on servers, and reduce the power consumptions in data centers. Another commonly used approach is to employ the dynamic voltage and frequency scaling (DVFS) to dynamically adjust the performance and therefore the power dissipation of a server to achieve power efficiency. For example, in [8], Beloglazov et al. propose a two-layer optimization (e.g. global and local layers) to minimize the power consumption in data centers. The local layer monitors each virtual machine's utilization, and dynamically adjusts a physical server's working frequency through dynamic voltage and frequency scaling (DVFS) [9]. The global layer uses bin-packing methods to find a new destination for the virtual machine that has to be migrated. Wang et al. [10] propose an approach for power minimization by determining the appropriate numbers of powered-on servers and their running modes (i.e. execution speeds) for the service requests, which can be modeled using the traditional queuing model.

Power saving techniques usually, if not always, lead to degraded computing performance. However, the quality of service is key to client satisfaction. Service providers usually provide multiple service level agreements (SLAs) regarding different QoS levels. A database may be queried internally by a company's employees or externally by the company's customers. The external customers may further be divided into various priority groups. All of the different "visitors" have their own QoS requirements. The challenges is then how to minimize the power consumption and guarantee these QoS requirements.

With the virtualization technology, it has been a common approach (e.g. [11]) to serve the requests with the same QoS on the same server. Since all requests on the same virtual machine has the same QoS requirement, different levels of QoS can be captured by one single variable such as resource provisions (e.g. [6][7][8]), required processing speeds (e.g. [9]) or latency (e.g. [10][11]). While this approach simplifies the resource management problem to guarantee one specified QoS requirements, it limits the requests that can share resources, and the overall resource usage can be rather inefficient, as shown later in this paper.

In this paper, we study the problem of how to minimize power consumption for data centers with guaranteed QoS levels. We assume that data centers are able to accommodate requests of different kinds of services, as well as requests of the same service but with different QoS requirements. Different from previous studies that employ an individual server (or virtual machine) for the requests with the same QoS, we develop a method that enables requests of the same service but with different QoS levels to share the same server, and therefore greatly increase the resource utilization. To our best knowledge, this is the first approach developed that can guarantee different levels of QoS for requests that share the same server. In addition, we devise a novel approach that can judiciously combine various types of requests to the same server, and properly discard potential failure requests in time to minimize total processing demands and thus power consumptions with statistical guaranteed QoS. Experimental results show that our proposed method has much better performance than other commonly used methods in terms of both QoS satisfactions and lower power consumptions.

II. Preliminary

In this section, we introduce our system model and formulate the problem.

Parameters	Definitions
n	total number of services.
$\tau_{i,j}$	the j-type requests in service S_i.
r_i	the total number of request types for service S_i.
$V_{i,k}$	the k-th server that supplies service S_i.
m_i	the total number of servers that support S_i service.
P^d	the dynamic power.
P^s	the static power.
$\lambda_{i,j}$	the arrival rate of $\tau_{i,j}$.
$D_{i,j}$	the deadline for $\tau_{i,j}$.
$R_{i,j}$	the completion ratio for $\tau_{i,j}$.
$Q_{i,j}$	the quality-of-service requirement of $\tau_{i,j}$.
$\mu_{i,j}$	the required processing rate for $\tau_{i,j}$ in order to guarantee $Q_{i,j}$.
$U_{i,k}$	the required processing rate of the k-th server to provide QoS-guaranteed service S_i.
Ω	the pool rate, the sum of all the servers' processing rate in a server pool. $\Omega = \sum_i \sum_k U_{i,k}$.
C	the capacity of the servers.

TABLE I
Parameter notation.

A. System model

We assume that a data center provides n different services based on their application purposes, i.e. $S = \{S_1, S_2, ..., S_n\}$. Each service S_i can accommodate different types of requests $\Gamma_i = \{\tau_{i,j}, j = 1, \ldots, r_i\}$, i.e. requests under different service level agreements. Each $\tau_{i,j}$ has its own QoS requirements, denoted as $Q_{i,j}$. We assume that different services need to be hosted on different servers but different types of requests in S_i can be potentially hosted in the same server. We assume that there are n types of servers, i.e. $\{V_1, V_2, ..., V_n\}$, and each type (e.g. V_i) may contain m_i servers (e.g. $V_{i,k}, k = 1, \ldots, m_i$) that support service S_i. Our system model is illustrated in Figure 1. In Figure 1, τ_{11} τ_{14} share server V_{11}, and τ_{12} τ_{13} share server V_{12}.

We assume that the request arrival patterns follow the Poisson distributions [12], and their response times follow exponential distributions. Different types of requests for the same service may have different arrival times, deadlines, and completion ratio requirements. Specifically, a request is modeled with a 3-tuple, i.e. $\tau_{ij} = \{\lambda_{i,j}, D_{i,j}, R_{i,j}\}$, where

- $\lambda_{i,j}$: the arrival rate of the j-type requests in service S_i.

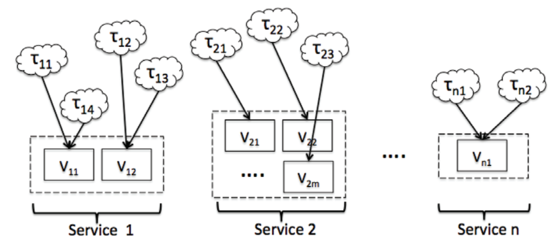


Fig. 1. System model.

- $D_{i,j}$: the deadline of the j-type requests in service S_i.
- $R_{i,j}$: the completion ratio requirement of the j-type requests in service S_i.

The $Q_{i,j}$ of $\tau_{i,j}$ is defined by $\{D_{i,j}, R_{i,j}\}$, meaning that at least $R_{i,j}$ percent of the $\tau_{i,j}$ requests have to be served no later than $D_{i,j}$.

A service provider suffers from high power consumptions while providing QoS guaranteed services. We adopt a linear power model to describe a server's power consumption (e.g. [7]), as shown in Equation (1):

$$P = P^d \varphi + P^s \tag{1}$$

where P^s is the static power, and $P^d \varphi$ is the dynamic power. φ is the utilization of a server, defined as $\varphi = \frac{U}{C}$, where U is the required processing rate of a server, and C is the capacity limit of a server. For ease of reference, we list some commonly used notations in Table I.

B. Problem definition

With the system model defined above, the problem we are to address can be formulated as follows.

Problem 1: Given service requests $\Gamma_i = \{\tau_{i,j} : j = 1, ..., r_i\}$ for $i = 1, \ldots, n$, determine the server pool, i.e. $V = \{V_{i,k} : i = 1, ..., n; k = 1, ..., m_i\}$, the corresponding processing rate $U_{i,k}$, and the allocation of Γ_i to V, such that the QoS requirements (i.e. $Q_{i,j}, i = 1, ..., n; j = 1, ..., r_i$) are guaranteed and the power consumption of the server pool is minimized.

This problem involves with two intertwined problems: how to judiciously pack service requests to a server and how to determine the proper service rate to minimize the power consumption while guaranteeing the QoS for all types of requests. In what follows, we first introduce three preliminary analyses, and then present our algorithm in details.

III. Preliminaries

This section presents three key analyses regarding QoS guarantee, request multiplexing, and request packing. These analyses form the basis of our approach.

A. Processing rate selection for QoS guarantee

Traditionally, people using M/M/1 queue [12] to represent the request processing procedure [13], as shown in Figure 2. Service requests arrive with a rate λ, wait in a queue with an infinite size, and get processed with a rate μ.

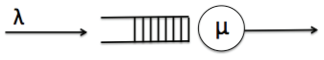


Fig. 2. M/M/1 queue.

Accordingly, the probability density function (PDF) and the cumulative distribution function (CDF) of the response time are listed below:

$$f(t) = (\mu - \lambda)e^{-(\mu-\lambda)t} \tag{2}$$

$$F(t) = 1 - e^{-t\mu(1-\frac{\lambda}{\mu})} \tag{3}$$

with a mean response time:

$$E[t] = \frac{1}{\mu - \lambda} \tag{4}$$

The q-percentile of the response time t_q (t_q is larger than $q\%$ of all response times) has the following relationship:

$$1 - e^{-t_q\mu(1-\frac{\lambda}{\mu})} = \frac{q}{100} \tag{5}$$

with simple transformations, we have:

$$\begin{aligned} t_q &= \frac{1}{\mu - \lambda} ln[\frac{100}{100 - q}] \\ &= E[t] ln[\frac{100}{100 - q}] \end{aligned} \tag{6}$$

Given request $\tau_{i,j}$'s arrival rate $\lambda_{i,j}$, deadline $D_{i,j}$, and completion ratio requirement $R_{i,j}$, in order to guarantee $Q_{i,j}$, the required service rate $\mu_{i,j}$ (when $\tau_{i,j}$ is hosted alone) is:

$$\mu_{i,j} = \frac{ln[\frac{1}{1-R_{i,j}}]}{D_{i,j}} + \lambda_{i,j} \tag{7}$$

While $\mu_{i,j}$ defined above can guarantee $Q_{i,j}$, as much as $(1 - R_{i,j})\%$ can miss their deadlines and contribute little or no benefit to the application at all. To save power consumption, it is desirable that we can discard a request if it has a high probability to miss its deadline but save the precious resource for requests that are more likely to be successfully fulfilled. The problem is how to discard these requests without compromising the QoS. To this end, we employ the renege $M/M/1$ queuing model [14], as illustrated in Figure 3.

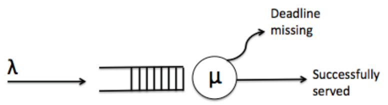


Fig. 3. M/M/1 with renege.

As shown in Figure 3, according to the renege model, each request is associated with a deadline. If a request is not fully served until its deadline, it is removed from the system. According to this model, there exists an interesting relationship among the request's deadline miss probability P_{miss}, the request arrival rate λ, processing rate μ, and deadline D, which can be formulated as:

$$P_{miss} = \frac{(1-\rho)e^{\mu D(\rho - 1)}}{1 - \rho e^{\mu D(\rho-1)}} \tag{8}$$

where $\rho = \frac{\lambda}{\mu}$. Accordingly, for the given $\lambda_{i,j}$, $R_{i,j}$, and $D_{i,j}$ of request $\tau_{i,j}$, we can derive $\mu^*_{i,j}$ that guarantees $Q_{i,j}$:

$$1 - R_{i,j} \geq \frac{(1 - \frac{\lambda_{i,j}}{\mu^*_{i,j}})e^{\mu^*_{i,j}D_{i,j}(\frac{\lambda_{i,j}}{\mu^*_{i,j}}-1)}}{1 - \frac{\lambda_{i,j}}{\mu^*_{i,j}}e^{\mu^*_{i,j}D_{i,j}(\frac{\lambda_{i,j}}{\mu^*_{i,j}}-1)}} \tag{9}$$

It is interesting to note that, the processing rate derived using renege queue $\mu^*_{i,j}$ is smaller than the $\mu_{i,j}$ in Equation (7) (because of page limit, we omit the proof). It means that, with the renege model and judiciously remove the request from the queue, we can guarantee the same QoS with smaller processing rates.

B. Request multiplexing

When service S_i has multiple types of requests, e.g. $\tau_{i,j}$, one common approach is to host each type of requests with one individual server or virtual machine. While this approach simplifies the resource management for guaranteeing specified QoS levels, the efficiency of resource usage can be rather low, not to mention other possible problems, such as the cost of software licenses.

Consider three types of requests of the same service τ_{11}, τ_{12} and τ_{13}. The $R_{i,j}$-th percentile response time $t_{R_{i,j}}$ of $\tau_{i,j}$ can be formulated as:

$$t_{R_{i,j}} = \frac{1}{\mu_{i,j} - \lambda_{i,j}} ln[\frac{1}{1 - R_{i,j}}] \tag{10}$$

When hosting each $\tau_{1,j}$ individually on a service pool $V = \{V_{1,1}, V_{1,2}, V_{1,3}\}$, we let the server processing rates as U_{11}, U_{12}, and U_{13}, respectively. Then, to satisfy each $Q_{i,j}$, the processing rates can be calculated as follows:

$$U_{i,j} = \mu_{i,j} = \frac{ln[\frac{1}{1-R_{i,j}}]}{D_{i,j}} + \lambda_{i,j} \tag{11}$$

Let us define *the processing rate of the server pool*[1] as the sum of all the servers' required processing rates in a server pool (i.e. V), denoted as $\Omega(V)$. Then, we have:

$$\Omega(V) = \sum_{j=1}^{3} \mu_{1,j} = \sum_{j=1}^{3} [\frac{ln[\frac{1}{1-R_{1,j}}]}{D_{1,j}} + \lambda_{1,j}] \tag{12}$$

When the three types of requests are hosted together in a single server $V' = V'_{1,1}$, the processing rate $U'_{1,1}$ of $V'_{1,1}$ has to satisfy the QoS requirements of $\tau_{1,1}$, $\tau_{1,2}$, and $\tau_{1,3}$. For example, assume that the required processing rate of $V'_{1,1}$ is $u^*_{1,1}$ according to $\tau_{1,1}$'s QoS requirements (i.e. $U'_{1,1} = u^*_{1,1}$). Then, based on Equation (10), we have

$$\frac{1}{\mu_{1,1} - \lambda_{1,1}} ln[\frac{1}{1 - R_{1,1}}] = \frac{1}{u^*_{1,1} - \sum_{j=1}^{3}\lambda_{1,j}} ln[\frac{1}{1 - R_{1,1}}] \tag{13}$$

We transform Equation (13) and have:

$$\begin{aligned} U'_{1,1} = u^*_{1,1} &= \mu_{1,1} - \lambda_{1,1} + \sum_{j=1}^{3} \lambda_{1,j} \\ &= \mu_{1,1} + \sum_{j=1, j\neq 1}^{3} \lambda_{1,j} \end{aligned} \tag{14}$$

The $U'_{1,1}$ derived by Equation (14) guarantees that after the combination, the processing of $\tau_{1,1}$ is the same as it is hosted separately by a single server. Similarly, we can derive the $V'_{1,1}$'s required processing rate $U'_{1,1} = u^*_{1,2}$ ($U'_{1,1} = u^*_{1,3}$, resp.) according to the QoS requirements of $\tau_{1,2}$ ($\tau_{1,3}$, resp.) as follows:

$$U'_{1,1} = u^*_{1,2} = \mu_{1,2} + \sum_{j=1, j\neq 2}^{3} \lambda_{1,j} \tag{15}$$

and

[1]The processing rate of the server pool, i.e. Ω, is a parameter for reflecting the overall physical server utilization in the server pool.

$$U'_{1,1} = u^*_{1,3} = \mu_{1,3} + \sum_{j=1,j\neq 3}^{3} \lambda_{1,j} \tag{16}$$

Therefore, when $\tau_{1,1}$, $\tau_{1,2}$, and $\tau_{1,3}$ are hosted together in $V'_{1,1}$, in order to satisfy the QoS requirements for all the three types of requests simultaneously, the processing rate $U'_{1,1}$ of $V'_{1,1}$ has to be the maximum among Equations (14), (15), and (16), i.e.:

$$U'_{1,1} = max\{u^*_{1,1}, u^*_{1,2}, u^*_{1,3}\} \tag{17}$$

Then, the server pool processing rate is $\Omega(V') = U'_{1,1}$.

We formally formulate the discussion above in Theorem 1 to conclude our analysis. The proof is omitted due to page limit.

Theorem 1: For service requests $\tau_{i,1}$, $\tau_{i,2}$, ..., τ_{i,r_i} hosted in a single server $V_{i,1}$, in order to satisfy the QoS requirements for $\tau_{i,j}, j = 1, 2, \ldots, r_i$, we need to have

$$U_{i,1} = u^*_{i,j} = \mu_{i,j} + \sum_{q=1,q\neq j}^{r_i} \lambda_{i,q} \tag{18}$$

Then $\tau_{i,1}$, $\tau_{i,2}$, ..., τ_{i,r_i} can all meet their QoS if $U_{i,1} \geq max\{\mu^*_{i,j}, j = 1, 2, \ldots, r_i\}$.

From Theorem 1, to meet QoS of $\tau_{1,1}$, $\tau_{1,2}$, and $\tau_{1,3}$ when multiplexing a server, we have $\Omega(V') = u^*_{1,\alpha} = \mu_{1,\alpha} + \sum_{j=1,j\neq\alpha}^{3} \lambda_{1,j}$ (assuming $u^*_{1,\alpha} = max\{u^*_{1,j}, j = 1, \ldots, 3\}$). Therefore, to compare $\Omega(V)$ and $\Omega(V')$, we have:

$$\frac{\Omega(V)}{\Omega(V')} = \frac{\sum_{j=1}^{3} \mu_{1,j}}{\mu_\alpha + \sum_{j=1,j\neq\alpha}^{3} \lambda_{1,j}} > 1 \tag{19}$$

which indicates that to guarantee the same QoS level, request multiplexing requires a smaller processing rate of the server pool (i.e. Ω) than hosting each type of requests in an individual server separately.

C. Request packing

From the discussions above, clustering multiple types of requests into the same server helps improve the resource usage. The question is then how to identify the group of request types for each server that can minimize the server pool processing rate Ω (i.e. $\Omega = \sum_i \sum_k U_{i,k}$) and thus maximize the resource usage efficiencies. Consider the following example with four types of requests $\tau_1 \ldots \tau_4$:

- $\lambda_1 = 60$, $\mu_1 = 120$.
- $\lambda_2 = 40$, $\mu_2 = 80$.
- $\lambda_3 = 50$, $\mu_3 = 70$.
- $\lambda_4 = 20$, $\mu_4 = 90$.

with μ_i the minimum processing rate to satisfy its QoS when allocated to a server individually. All parameters have the unit of $(request/second)$. With the two request combinations shown in Figure 4, in order to guarantee all the QoS requirements, the required server pool processing rates are $\Omega(V_1, V_2) = 300$ and $\Omega(V'_1, V'_2) = 280$ (derived based on Equation (17)). This example clearly shows that different service request allocations can lead to server pools with very different processing rates.

The service request allocation problem, as we proposed to study here, is $\mathcal{NP}$-hard in nature. Therefore, we focus on the development of effective and efficient heuristic solution for this problem. To this end, we have made a number of interesting observations, which are formulated in the following theorems. Due to page limits, we omit the proofs.

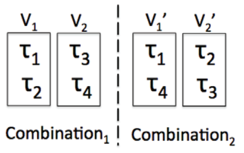


Fig. 4. Processing rate with different combination.

Theorem 2: Let $\Gamma_{1,p}$ and $\Gamma_{1,q}$ be two sets of requests of the same service mapped to two servers $V_{1,p}$ and $V_{1,q}$, respectively. Assume that one request type, i.e. $\tau_{1,i} \in \Gamma_{1,p}$, is migrated from $V_{1,p}$ to $V_{1,q}$. Let $U_{1,p}$ ($U_{1,q}$, resp.) be the minimum processing rate for $V_{1,p}$ ($V_{1,q}$, resp.) that guarantees the QoS requirements for requests allocated to the server after $\tau_{1,i}$ is migrated. Then the processing rate for the server pool, i.e. $\Omega(V) = U_{1,p} + U_{1,q}$, is minimized when $\phi_{i,1} = \mu_{i,1} - \lambda_{i,1}$ is the minimum.

Theorem 2 implies that when migrating requests from one server to another, selecting the requests with the smallest ϕ helps reduce the overall server pool processing rate. In addition, we have the following theorem.

Theorem 3: Let $V_1 = \{V_{1,1}, V_{1,2}, \ldots, V_{1,p}\}$ and $V'_1 = \{V'_{1,1}, V'_{1,2}, \ldots, V'_{1,q}\}$ be two server pools that host the same set of requests Γ_1. Both V_1 and V'_1 satisfy Γ_1's QoS requirements. Then $\Omega(V_1) \geq \Omega(V'_1)$ if $p \geq q$.

Theorem 3 indicates that for a server pool that holds the same set of requests, the smaller the number of server is, the smaller the processing rate it needs to guarantee all the QoS requirements. Therefore, to improve the resource usage and minimize the power consumption, it is beneficial to reduce the number of servers as best as we can.

IV. The request allocation algorithm

We are now ready to discuss our approach for power consumption minimization in data centers with guaranteed QoS.

Since the overall power consumption depends on both the processing rate of the server pool and their static power consumptions (see Equation (1)), to solve Problem 1, we need to minimize the number of employed servers and their utilizations. As discussed above, when multiple requests are hosted in the same server, the processing rate of the server pool can be greatly reduced. Also, when requests with long latency and high possibility of missing its deadline can be judiciously removed, the demanded server processing rate can also be reduced. Therefore, in our approach, we adopt the renege model to enable the guarantee of different QoS levels for requests that share the same server, and also save the power consumption by expunging the requests that have high probability to fail. In the meantime, Theorems 2 and 3 provide valuable insights for the development of heuristic to minimize the number of servers as well as the processing rate of the server pool. The detailed algorithm is illustrated in Algorithm 1.

As shown in Algorithm 1, the required processing rate μ_j for each type of requests $\Gamma = \{\tau_j, j = 1, \ldots, r\}$ is calculated using the renege model (line 1). We then sort all requests based on $\phi_j, j = 1, \ldots, r$ in a decreasing order (line 2), with $\phi_j = \mu_j - \lambda_j$, where λ_j is the request τ_j's arrival rate, and μ_j is the required processing rate of a server to satisfy Q_j if τ_j is hosted alone. Then, we employ the traditional first-fit bin-packing algorithm to pack requests in the list to servers with a capacity (i.e. the maximum processing rate) of C (line 3 to line 13). The reason to order

Algorithm 1: Request allocation

Input : A set of requests $\Gamma = \{\tau_j, j = 1, \ldots, r\}$ in service S, each τ_j has corresponding λ_j and Q_j ($\{D_j, R_j\}$). A set of servers $V = \{V_k, k = 1, \ldots, m\}$ with the same capacity C serve the requests in service S.

Output: The requests allocation.

```
Calculate each $\mu_j$ to satisfy $Q_j$ based on Equation (9);
$dif\_vect \leftarrow$ Sort the $\phi = \mu - \lambda$ in the decreasing order;
for All requests $\tau_j$, $j = 1, 2, \ldots, r$ do
    for All servers $V_k$, $k = 1, 2, \ldots, m$ do
        Add $\tau_j$ into server $V_k$;
        Calculate $V_k$'s required processing rate $U_k$ based on Equation (17);
        if $U_k \leq C$ then
            Remove $\tau_j$ from $dif\_vect$;
            Break the loop and pack the next request;
        else
            Remove $\tau_j$ from the current server $V_k$;
    if Request $\tau_j$ did not fit in any available server then
        Open a new server and pack request $\tau_j$;
```

the requests according to the value of ϕ is because, according to Theorem 2, when a server is full and new server needs to be powered-on, allocating the requests with the smallest value of ϕ helps reduce the overall processing rate of the server pool. Furthermore, the bin-packing algorithm minimizes the "bins", or the number of servers. It thus also minimizes the processing rate of the server pool as shown in Theorem 3. Therefore, Algorithm 1 can potentially achieve high resource usage efficiencies (for both computing and energy resources).

V. Experimental results

In this section, we use simulations to study the properties of our proposed approach.

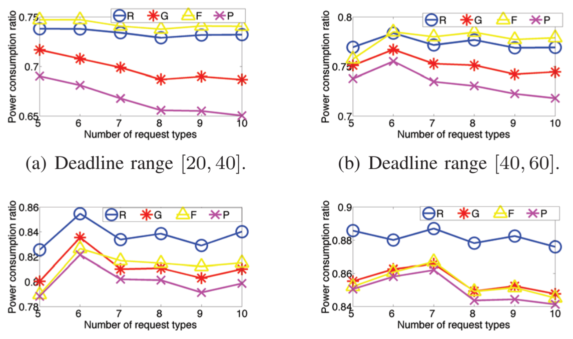
We assume that requests for different services must be assigned to different servers. In our experiments, all requests are of the same service but with different QoS levels. The arrival rate of each request type was randomly generated following a uniform distribution in a range between 120 $requests/second$ and 20 $requests/second$. The deadline of each request type was also randomly generated following a uniform distribution in a range between $100ms$ and $80ms$. The completion ratio was fixed at 95%. The server's capacity was set to be 250 $requests/second$.

We compare our approach to four other approaches. All of them apply renege model to derive the required processing rate for each type of requests.

- **split**: denoted as "S", the traditional method that each request type is hosted in an isolated server [6][15].
- **random**: denoted as "R", a method that combines the requests randomly using request multiplexing.
- **first-fit-decreasing**: denoted as "F", a widely employed bin-packing method [16][17]. Requests are combined using request multiplexing in a decreasing order of $\mu_{i,j}$.
- **greedy**: denoted as "G", a greedy packing approach. It packs all the requests into one server, e.g. V_1, using request multiplexing. If the required processing rate of the server exceeds the server's capacity, the request type that has the smallest ϕ will be migrated into a new server, e.g. V_2. The migration continues until V_1's required processing rate U_1 is smaller than or equals to the capacity C, i.e. $U_1 \leq C$. Then, the same procedure works in V_2 and stops when all the servers, e.g. V_1, ..., V_m satisfy their capacity constraints.
- **Proposed**: denoted as "P", our proposed method.

A. Service utilization effect

We first conducted a set of tests to study the power saving performance by different approaches under different request utilizations, i.e. $\rho_{i,j} = \frac{\lambda_{i,j}}{\mu_{i,j}}$. From Equation (9), when the arrival rate is a constant value, the smaller the deadline, the higher the required processing rate is. Therefore, as the deadline $D_{i,j}$ reduces, the processing rate $\mu_{i,j}$ increases, and then the service utilization increases. We therefore varied request utilization by changing the interval from which we randomly picked the deadlines. Specifically, we modified the upper bound and lower bound of the interval simultaneously, from $100ms$ to $40ms$ and from $80ms$ to $20ms$, respectively, with an interval length of $20ms$. The test cases in each intervals was tested a thousand times. The averaged power consumption results were collected, normalized to that by the approach "split", and are shown in Figure 5.



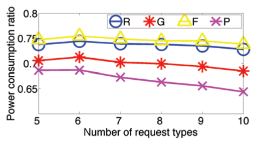
(a) Deadline range $[20, 40]$.
(b) Deadline range $[40, 60]$.
(c) Deadline range $[60, 80]$.
(d) Deadline range $[80, 100]$.

Fig. 5. Power saving performance for requests with different deadline ranges [lower bound, upper bound].

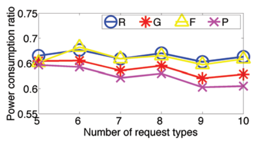
As shown in Figures 5(a) to 5(d), our proposed method (denoted as "P") has the lowest power consumption ratio across all experiment settings. The figures clearly illustrate the fact that the power consumption ratio decreases as the utilization becomes higher (which indicates an increasing resource saving). When the number of request types is 10, the power consumption ratio achieved by our proposed approach increased from 65% to 85% as the utilization increases (e.g. the deadline range changes from [$20ms$, $40ms$] to [$80ms$, $100ms$]). Since the "greedy" approach uses a similar allocation strategy that is based on the sorting result of $\phi_{i,j}$, its power saving performance is worse than but very close to the results achieved by our proposed approach. The "first-fit-decreasing" (ffd) derives the request allocations according to the sorting of $\mu_{i,j}$, which does not have direct effects on the processing rate of the server pool (i.e. Ω) and thus has a poor power saving performance, especially when the utilization is high (as shown in Figure 5(a)). However, when the utilization is low (as shown in Figure 5(d)), "ffd" has a similar performance as our proposed approach because the sorting of $\phi_{i,j}$ is similar to the sorting of $\mu_{i,j}$.

B. Capacity effect

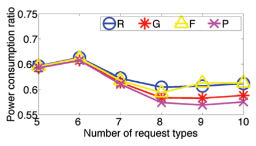
Next, we study the server's capacity effect on power savings. In this experiment, we kept the request deadline's upper and lower bounds at $40ms$ and $20ms$, and gradually changed the server's capacity from 250 $requests/second$ to 550 $requests/second$ with an interval length of 100 $requests/second$. The completion ratios for all requests were set at 95%. We ran each setting a thousand times.



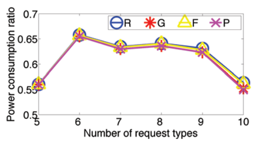
(a) $C = 250\ requests/second$



(b) $C = 350\ requests/second$



(c) $C = 450\ requests/second$



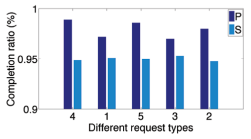
(d) $C = 550\ requests/second$

Fig. 6. Server's capacity effects on power saving.

The averaged results are shown in Figure 6. The server's capacity increases from 250 $requests/second$ in Figure 6(a) to 550 $requests/second$ in Figure 6(d). Our proposed approach outperforms the others throughout all of the test settings. However, the improvement of our proposed approach diminishes as the server's capacity increases. This is because when the server's capacity increases, more types of requests can be hosted together in the same server. All the approaches that employ request multiplexing can achieve much better energy saving results than that when each request type is hosted solely on an individual server. If the capacity is large enough that all the requests can be packed into a single server, then all the approaches that employ request multiplexing have the same results.

C. Completion ratio and average response time

Finally, we compare the request completion ratios and average response times achieved by our proposed approach and the split method. The test was conducted with 5 request types. The deadline's upper and lower bounds were set to $40ms$ and $20ms$. The arrival rate's upper and lower bounds were set as 120 $requests/second$ and 20 $requests/second$. Completion ratios were set to 95% for all requests. Our proposed method provided a combination, indicating that $\{\tau_4, \tau_1, \tau_5\}$ and $\{\tau_3, \tau_2\}$ were hosted separately in two servers. Each request type had $10,000$ requests. The averaged simulation results are shown in Figure 7.



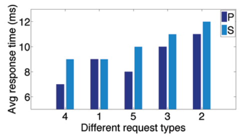


Fig. 7. Completion ratios and average response times.

As reflected in Figure 7(a), our proposed method not only guarantees the completion ratios (actually our method achieves even higher completion ratios), but also obviously reduces the average response times as shown in Figure 7(b). The reason for this phenomenal improvement is because of the request multiplexing technique, which efficiently utilizes computing resources among different types of requests.

VI. Conclusion

The expansion of web services in both scope and scale make efficient resource management extremely challenging for a service provider's sustainable development. In this paper, we propose an approach based on request multiplexing and renege queueing techniques to judiciously combine different types of requests for each server and discard potential failure requests in time. A data center's power consumption can be reduced significantly without compromising QoS satisfactions.

VII. Acknowledgement

This work is supported in part by NSF under projects CNS-1423137 and CNS-1018108.

References

[1] Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. Cutting the electric bill for internet-scale systems. *SIGCOMM Comput. Commun. Rev.*, 39(4):123–134, August 2009.
[2] Luis Toms and Johan Tordsson. Improving cloud infrastructure utilization through overbooking. In Salim Hariri and Alan Sill, editors, *CAC*, page 5. ACM, 2013.
[3] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Y H. Katz, and Michael A. Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis, 2012.
[4] Pocket gems on google cloud platform. http://cloud.google.com/customers/pocketgems/.
[5] Sony music on google cloud platform. http://cloud.google.com/customers/sony-music/.
[6] Akshat Verma, Gargi Dasgupta, Tapan Kumar Nayak, Pradipta De, and Ravi Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, pages 28–28, Berkeley, CA, USA, 2009. USENIX Association.
[7] Carlo Mastroianni, Michela Meo, and Giuseppe Papuzzo. Probabilistic consolidation of virtual machines in self-organizing cloud data centers. *IEEE T. Cloud Computing*, 1(2):215–228, 2013.
[8] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *CCGRID*, pages 826–831. IEEE, 2010.
[9] Gregor von Laszewski, Lizhe Wang, Andrew J. Younge, and Xi He. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *CLUSTER*, pages 1–10. IEEE, 2009.
[10] Shengquan Wang, Waqaas Munawar, Jun Liu, Jian-Jia Chen, and Xue Liu. Power-saving design for server farms with response time percentile guarantees. In Marco Di Natale, editor, *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 273–284. IEEE, 2012.
[11] Palden Lama and Xiaobo Zhou. Efficient server provisioning with end-to-end delay guarantee on multi-tier clusters. In *IWQoS*, pages 1–9. IEEE, 2009.
[12] T.G. Robertazzi. *Computer Networks and Systems: Queueing Theory and Performance Evaluation*. Telecommunication networks and computer systems. Springer, 2000.
[13] M. Zivkovic, J.W. Bosman, J. L. Van den Berg, R.D. Van der Mei, H.B. Meeuwissen, and R. Nunez-Queija. Dynamic profit optimization of composite web services with slas. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–6, Dec 2011.
[14] D. Y. Barrer. Queuing with impatient customers and ordered service. *Operations Research*, 5(5):pp. 650–656, 1957.
[15] E. Feller, L. Rilling, and C. Morin. Energy-aware ant colony based workload placement in clouds. In *Grid Computing (GRID), 2011 12th IEEE/ACM International Conference on*, pages 26–33, Sept 2011.
[16] Xiaoqiao Meng, Canturk Isci, Jeffrey O. Kephart, Li Zhang, Eric Bouillet, and Dimitrios E. Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In Manish Parashar, Renato J. Figueiredo, and Emre Kiciman, editors, *ICAC*, pages 11–20. ACM, 2010.
[17] Yasuhiro Ajiro and Atsuhiro Tanaka. Improving packing algorithms for server consolidation. In *Int. CMG Conference*, pages 399–406. Computer Measurement Group, 2007.