TailCon: Power-Minimizing Tail Percentile Control of Response Time in Server Clusters

Xi Chen*, Xue Liu*, Shengquan Wang†, Xiao-Wen Chang*
* School of Computer Science, McGill University,
Email: xi.chen7@mail.mcgill.ca, {xueliu,chang}@cs.mcgill.ca
‡ Department of Computer and Information Science, University of Michigan - Dearborn
Email: {shqwang}@umich.edu

***Abstract*—To provide satisfactory customer experience, modern server clusters like Amazon usually set Service Level Agreement (SLA) as guaranteeing a certain percentile (i.e. 99%) of the customer requests to have a response time within a threshold (i.e. $1s$). One way to meet the SLA constraint is to serve the customer requests with sufficient computing capacity based on the worst-case workload estimation in the server cluster. However, this may cause unnecessary power consumption in the server cluster due to over-provision of the computing capacity especially when the workload is highly dynamic. In this paper, we propose an adaptive computing capacity allocation scheme referred to as TailCon. TailCon aims at minimizing the power consumption in the server cluster while satisfying the SLA constraint by adjusting the number of active servers and the CPU frequencies of the turn-on machines online. In TailCon, we analyze the distribution of the request response time dynamically and leverage the measured request response time to estimate the workload intensity in the server cluster, which is used as a continuous feedback to find the proper provision of the computing capacity online based on optimization techniques. We conduct both the emulation using the real-word HTTP traces and the experiments to evaluate the performance of TailCon. The experimental results demonstrate the effectiveness of TailCon scheme in enforcing the SLA constraint while saving the power consumption.**

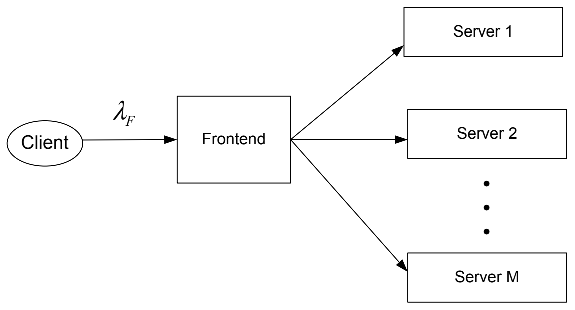
I. Introduction

Power consumption becomes a major concern in modern server clusters. With the growth of user demands to the online commercial services, the service providers invest enormously in establishing and maintaining the server clusters to server the on-demand workload [1]. The operation of the servers and the cooling systems in the server clusters consume huge amount of power. Statistics show that the electricity bills for maintaining the worldwide server clusters mount up to \$30 billion [2]. Since there are power-aware techniques like DVFS (dynamic voltage and frequency scaling) and DPM (dynamic power management) developed and applied to computers, it is highly desired to leverage these techniques for power management in the server clusters so that the power consumption can be reduced.

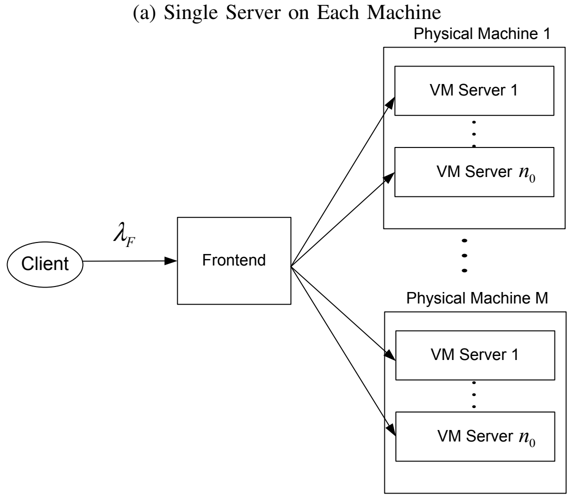
In the commercial server clusters, Quality of Service (QoS) is the primary concern to the service providers, which is usually evaluated by the completion of Service Level Agreement (SLA) contracted between the service provider and the clients. Failing to fulfill SLA will degrade QoS and further causes the financial losses to the service provider. SLAs are usually chosen to ensure that most online commercial clients can enjoy the service within a time threshold. Based on enterprize cost-benefit analysis, modern server clusters like Amazon [3] set their SLAs as to ensure that 99% of client requests can receive replies within a response time threshold. To provide the percentile guarantee of the response time, the service provider can allocate sufficient computing capacity to serve the incoming client requests in the server clusters according to the peak workload. However, the provision of the computing capacity based on the worst-case workload estimation may cause unnecessary power consumption especially when the workload is highly dynamic. How to provide satisfactory user experience with minimum power consumption is challenging.

In this paper, we provide a power-minimizing response time percentile control scheme, referred to as TailCon. The goal of TailCon is to minimize the power consumption in the server cluster while providing the percentile guarantee of the response time. The goal is met by dynamically providing the computing capacity in the server cluster to serve the incoming client requests, and the computing capacity includes the number of the active servers and CPU frequencies of the servers. In Tailcon, we analyze the statistics of the request response time and the power consumption of the server cluster based on real implementation. In the server cluster, the heavy-tailed distribution like Pareto distribution can well approximate the statistics of the requested webpage sizes [4]–[7]. We establish a server cluster with Apache servers, where each server serves the client requests with dynamic PhP webpages and the webpage sizes follow Pareto distribution. Experimental results show that the tail of the request response also follows Pareto distribution, where the lower bound of the request response time in the Pareto distribution is inversely proportional to the number of active servers and the CPU frequencies of the servers. In modern server clusters, a physical machine can either be configured as a single server or as a host holding several virtual machines with each virtual machine as an independent server, referred to as VM server. We use a physical machine with the two configurations to test the power consumption. We find that the power consumption of the physical machine has a cubic relation with the CPU frequencies in both configurations. Moreover, we do experiments to test the relation between the distribution of the request response time and the provision of the computing capacity, and build the system model accordingly. To adapt to the variation of the workload, we use the request response time as an online feedback to identify the parameters in the system model. Based on the system models, we design algorithms to minimize the power consumption with the SLA constraint by adjusting the provision of the computing capacity online. The contribution of the paper is summarized as follows:

- We design TailCon by considering multiple real-world server cluster configurations, i.e. a physical machine can be either a single server or a host for several VM servers. For each configuration, we provide the online computing capacity allocation scheme.
- We integrate multiple strategies to design the online capacity provision algorithm in TailCon, including capacity planning based on workload pattern identification, capacity pre-allocation based on workload change point

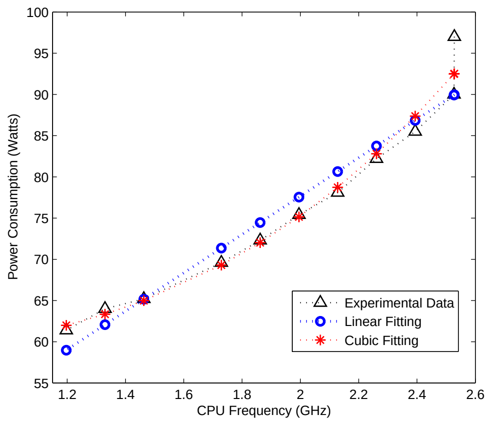


(a) Single Server on Each Machine

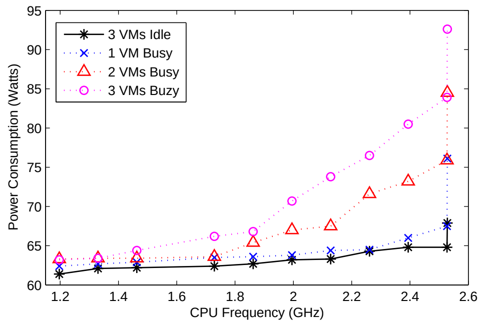


(b) Multiple VM Servers on Each Machine

Fig. 1: Server Cluster Configuration



(a) Single Server on Each Machine



(b) Multiple VM Servers on Each Machine

Fig. 2: Power Consumption with Different CPU Frequency of a Physical Machine

detection, and the capacity adjustment based on request response time feedback.

We implement both emulation and experiment to test the effectiveness of TailCon in different server cluster configurations, where we generate the client requests following the real-world HTTP trace [8]. Experimental results demonstrate the effectiveness of TailCon in meeting the SLA requirement while consuming much less power in comparison with a Baseline scheme.

The rest of the paper is organized as follows: We present a detailed analysis of power consumption and response time distribution in Section II. In Section III, we formulate the constraint optimization problem to minimize the power consumption subject to the SLA constraint. Section IV gives an overview of TailCon scheme. Section V and Section VI provide the experiment setup and results respectively. Section VII briefly surveys the related work. Section VIII concludes the paper.

II. System Model

In this section, we provide the structure of the server cluster and analyze the power consumption of the server cluster as well as the response time distribution.

A. Server Cluster Configuration

There are typically two ways to configure the modern server cluster: one way is to configure the physical machine as a single server, as shown in Fig. 1(a), and the other way is to configure the physical machine as a host that hosts n_0 servers, each of which is located on a VM, as shown in Fig. 1(b). Since the client requests are usually independently generated, we assume that the client requests arrive at the frontend in the server cluster following a Poisson distribution with a mean arrival rate λ^F [9], [10]. The frontend dispatches the incoming requests to the backend servers according to a load-balancing policy. In our paper, we assume all the servers in the cluster (either the physical machines in Fig. 1(a) or the VMs in Fig. 1(b) are homogeneous, hence round-robin is the optimal load-balancing policy. The reply to the request will be sent by the backend server and relayed by the frontend to the client. The workload in the server cluster can be highly dynamic, i.e. the arrival rate of the client requests may change from time to time during the runtime. Depending on the current workload intensity, the server cluster can allocate proper amount of computing capacity to serve the requests. The computing capacity in a server cluster is referred to as the number of active servers and the CPU frequencies of the machines holding the active servers. Given a server cluster with M physical machines, either configured as M servers or configured as M physical machines with each machine holding n_0 VM servers, we need to decide how many servers (or VM servers) should be activated on these physical machines and what the CPU frequency of the machines should be so that a given percentile of the client requests can receive the replies within a time threshold, and at the same time the power consumption is minimized.

B. Power Consumption of Server Cluster

We tested the power consumption of a physical machine with 2 Quad-core Intel Xeon processors and 4 GB

RAM running Centos 6.2 operating system. The physical machine supports DVFS, where the CPU frequency f can be adjusted in the range $\mathcal{F} = \{$1.197,1,330, 1.463,1.729,1.862,1.995,2.128,2.394,2.527,2.528$\}$GHz. When the machine is configured as a single server, the power consumption at different CPU frequencies when the server is fully loaded is shown in Fig. 2(a). The power consumption P of the machine at the busy state is

$$P^b = P^I + \alpha f^\theta, \tag{1}$$

where P^b, P^I are the power consumption of the server at the busy state and at the idle state respectively. We use $\theta = 1$ and $\theta = 3$ to approximate the relation between P^b and f respectively, and $\theta = 3$ is a better approximation as shown in Fig 2, which also matches the power model proposed in [11], [12]. In our model $P^I = 58.3648$Watts, $\alpha = 2.1143$Watts/GHz^3, and $\theta = 3$. When the machine is visualized using VirtualBox and holds multiple VM servers, we let the machine run at different CPU frequencies and in each frequency we test the power consumption of the machine by activating different VM servers. The power consumption is shown in Fig. 2(b). Note that for each active VM, we let them run at the full load in our test. The power consumption P can be modeled as a function of the CPU frequency f and the number of active VM servers n,

$$P^b = P^I + (\beta_0 + \beta_1 n) f^3, \tag{2}$$

where $P^I = 59.6221$, $\beta_0 = 0.2078$, $\beta_1 = 0.4892$, $n \leq n_0$ and $n_0 = 3$ in our configuration.

For a server, we usually consider the expected power consumption during a running period. Suppose the CPU utilization of the server is ρ, the expected power consumption $\mathbb{E}(P)$ is

$$\mathbb{E}(P) = (1 - \rho)P^I + \rho P^b, \tag{3}$$

and the expected power consumption of the server cluster, $\mathtt{E}(P^t)$, is

$$\mathbb{E}(P^t) = m\mathbb{E}(P) = mP^I + m\rho\alpha f^3, \tag{4}$$

in the first configuration and

$$\mathbb{E}(P^t) = m\mathbb{E}(P) = mP^I + m\rho(\beta_0 + \beta_1 n) f^3, \tag{5}$$

in the second configuration, where m is the number of turn-on machines and n is the number of active VM servers on each machine. Our goal is to minimize the expected power consumption in the server cluster $\mathbb{E}(P^t)$ by adjusting m, n and f dynamically subject to the percentile constraint of the request response time.

C. Response Time Distribution

The percentile constraint of the request response time, also known as SLA, is illustrated as follows. Given a server cluster running for a long period, the percentile of the request response time that exceeds a time threshold τ, referred to as the tail percentile, should be no larger than a percentile bound ξ. To meet the percentile constraint, a direct way is to measure the tail percentile online and adjust the provision of the computing capacity (usually chosen as m, n and f) accordingly. However, capacity allocation based on the measurement to the tail percentile may suffer from control saturation, i.e. when the measured tail percentile stays at 0 or 1, we can hardly estimate the intensity of the workload but only know the system is under-utilized or overloaded. In this case, it is hard to decide how much computing capacity should be provided to match the workload intensity. To avoid the control saturation, we would like to study the distribution of the response time R. Therefore, we can reinterpret the percentile constraint as controlling the complementary Cumulative Distribution Function (CDF) of the response time above the threshold τ, $\mathbb{P}(R > \tau)$, to stay under the percentile setpoint ξ,

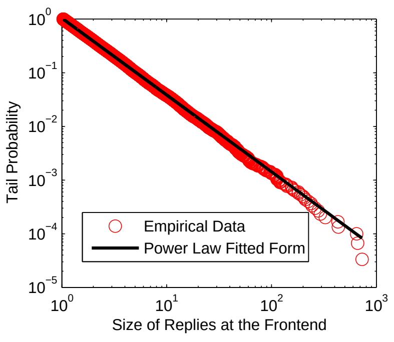
$$\mathbb{P}(R > \tau) \leq \xi. \tag{6}$$

The statistics of the request response time is closely related to the reply sizes of the request. It is shown that the file sizes of the web requests follow heavy-tailed distributions [13]–[15] and Pareto distribution in particular [9], [15]. We implement a server cluster with 6 Apache servers serving dynamic PhP webpages to the incoming requests, where the client requests are generated following Possion distribution with mean arrival rate λ_F and the PhP webpages are generated in a way so that their file sizes X follows Pareto distribution [16],

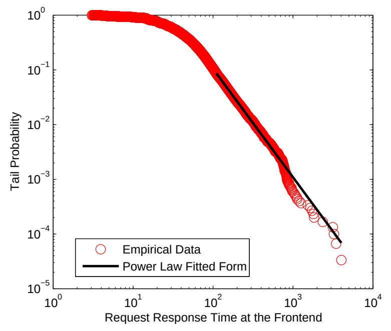
$$\mathbb{P}\{X \geq x\} = \begin{cases} (\frac{x}{\hat{x}})^{-\nu}, & x \geq \hat{x}, \\ 1, & x < \hat{x}, \end{cases} \tag{7}$$

where ν is the Pareto index, $1 < \nu < 2$, and $\hat{x}$ is the lower bound of the file size in Pareto distribution. By changing the distribution parameters in the file size distribution, like ν, $\hat{x}$, and by changing the request arrival rate λ_F, we tested the request response time in different situations, where each test involves 30000 samples of the request response time. In each test scenario, we use power-law fitting tool [17] to check whether the response time follows Pareto distribution, and if follows, what the distribution parameters are.

Fig. 3 and Fig. 4 show the power-law fitting result to the empirical data of the reply sizes and the response time of the client requests in two different test scenarios:



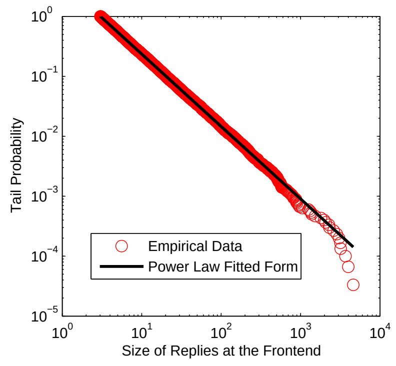
(a) Reply Size X Distribution



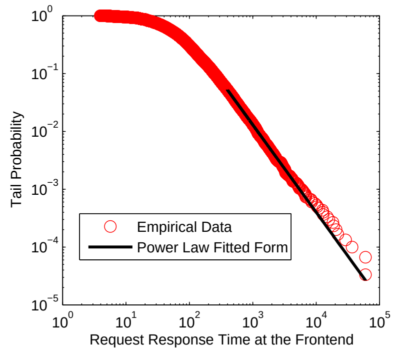
(b) Response Time R Distribution

Fig. 3: The Power-Law Fitting Results in Scenario 1

- **Scenario 1:** We set the mean request arrival rate $\lambda_F =$ 50 req/s. Each client request requiring a PhP webpage.



(a) Reply Size X Distribution



(b) Response Time R Distribution

Fig. 4: The Power-Law Fitting Results in Scenario 2

The sizes of the required PhP webpages follows Pareto distribution with $\nu = 1.4$ and $\hat{x} = 1$ KB. Then we get the response times of the client requests and the file sizes of the replies to the client requests at the frontend, and we use the power-law fitting tool to fit the empirical data, result is shown in Fig. 3.

- **Scenario 2:** We set the mean request arrival rate $\lambda_F = 40$ req/s. Each client request requiring a PhP webpage. The sizes of the required PhP webpages follows Pareto distribution with $\nu = 1.2$ and $\hat{x} = 3$ KB. Then we get the response times of the client requests and the file sizes of the replies to the client requests at the frontend, and we use the power-law fitting tool to fit the empirical data, result is shown in Fig. 3.

From Fig. 3 and Fig. 4, we see that the tail of the response time, starting from a point $\hat{y}$, follows Pareto distribution,

$$\mathbb{P}\{R > y\} = \left\{ \begin{array}{ll} (\frac{y}{\hat{y}})^{-\nu}, & y \geq \hat{y}, \\ 1, & y < \hat{y}, \end{array} \right. \tag{8}$$

where $\hat{y}$ is the lower bound of the response time in the Pareto distribution and ν is the Pareto index. The distribution parameters $\hat{y}$, ν, and the goodness-of-fitting ratio p in each test scenario is summarized in Table I. p is a parameter representing the goodness-of-fitting, where $p > 0.1$ indicates the empirical data matches Pareto distribution. Detailed explanation of p is omitted in this paper due to space limitation but can be referred to [17]. From Table I, we can see that in different test scenarios, when the reply size follows Pareto distribution, the tail of the response time also follows Pareto distribution with the goodness-of-fitting ratio $p > 0.1$.

The distribution of the response time can be influenced by the provision of the computing capacity, namely the number of

TABLE I: Distribution Parameter of Response Time in Different Test Scenarios

Parameters in X Distribution and λ_F	Parameters in R Distribution
$\nu = 1.2$,$\hat{x} = 1$KB,$\lambda_F = 50$req/s	$\nu = 1.6$,$\hat{y} = 73.83$ms,$p = 0.1224$
$\nu = 1.4$,$\hat{x} = 1$KB,$\lambda_F = 50$req/s	$\nu = 1.98$,$\hat{y} = 110.61$ms,$p = 0.6364$
$\nu = 1.6$,$\hat{x} = 1$KB,$\lambda_F = 50$req/s	$\nu = 1.98$,$\hat{y} = 65.35$ms,$p = 0.7213$
$\nu = 1.8$,$\hat{x} = 1$KB,$\lambda_F = 50$req/s	$\nu = 1.99$,$\hat{y} = 34.29$ms,$p = 0.8421$
$\nu = 1.2$,$\hat{x} = 3$KB,$\lambda_F = 40$req/s	$\nu = 1.51$,$\hat{y} = 393.52$ms,$p = 0.8062$
$\nu = 1.4$,$\hat{x} = 3$KB,$\lambda_F = 40$req/s	$\nu = 1.79$,$\hat{y} = 372.89$ms,$p = 0.7143$
$\nu = 1.6$,$\hat{x} = 3$KB,$\lambda_F = 40$req/s	$\nu = 1.73$,$\hat{y} = 95.84$ms,$p = 0.1051$
$\nu = 1.8$,$\hat{x} = 3$KB,$\lambda_F = 40$req/s	$\nu = 1.91$,$\hat{y} = 98.43$ms,$p = 0.1176$

active servers and their CPU frequencies. By fixing the number of active servers at 6 and changing the CPU frequencies of the turn-on machines, we find $\hat{y}$ in the Pareto distribution of response time R is inversely proportional to CPU frequencies when serving the same workload, as shown in Fig. 5(a). The same workload means the client requests in these test scenarios have the same arrival rate and the required webpage sizes follow the same Pareto distribution. Similarly, by fixing the CPU frequencies of the servers at their highest value 2.58GHz, we find $\hat{y}$ in the Pareto distribution of response time R is inversely proportional to the number of active servers, as shown in Fig. 5(b). We formulate $\hat{y}$ as a function of the number of active servers, m, and the CPU frequency of each active server f,

$$\frac{1}{\hat{y}} = cmf, \tag{9}$$

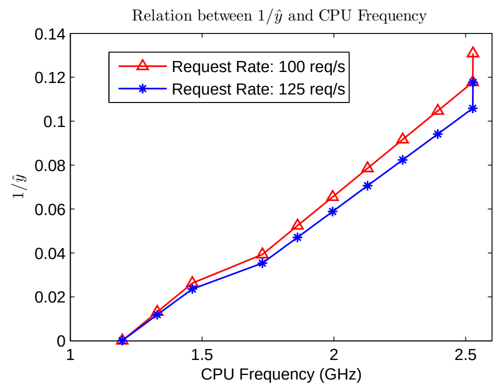
where the coefficient c can be influenced by the variation of the workload like request arrival rate λ_F, the distribution parameter of the reply sizes ν and $\hat{x}$. Since c is an unknown variable and can vary with the fluctuation of the workload, we need to estimate c online to acquire the accurate knowledge about the relation between the distribution of R and the provision of the computing resources. Note that Eq.(9) establishes only when each server in the cluster is not overloaded (CPU utilization $\rho = 1$) or severely under-utilized (CPU utilization $\rho \approx 0$).

From the Pareto distribution in Eq.(8) and Eq.(9), The response time percentile constraint (SLA requirement) in Eq.(6) can be rewritten as

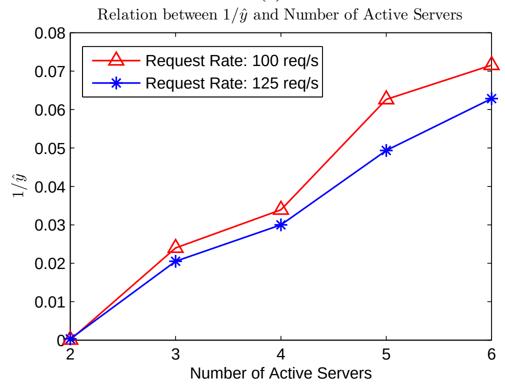
$$\frac{1}{\hat{y}} = cmf \geq \frac{1}{\tau \xi^{1/\nu}}. \tag{10}$$

III. Problem Formulation

Our goal is to minimize the expected power consumption in the server cluster while satisfying the SLA requirement, which can be achieved by adjusting the number of active servers and their CPU frequencies. Since the computing capacity provision is implemented online, we need to choose a sampling period. The sampling period is usually chosen short enough to ensure the workload stay relative steady during the period and long enough to serve a large number of client requests so that we can get enough request reply samples to analyze the response time distribution. In our paper, we choose the sampling period as the time interval to serve a fixed number of requests, i.e. 10000. After serving a fixed number of requests (like 10000 requests), we will adjust the computing capacity



(a)



(b)

Fig. 5: Relation Between $\frac{1}{\hat{y}}$ and the Provision of Computing Resources

provision in the cluster. We use k as the index of the sampling period, $k = 1, 2, ...,$ and the variables c, m, f etc. in the kth sampling period are denoted as c_k, m_k, f_k etc. Since there are two different configurations of the servers in the cluster, we formulate our goal as an optimization problem for each configuration.

A. One Server on One Physical Machine

Suppose each physical machine in the cluster is configured as a single server, the expected power consumption of the cluster is in Eq.(4) while the SLA constraint is in Eq.(10). Our goal can be formulated as the following problem

$$\begin{aligned} \min_{m_k, f_k} \; & P^I m_k + \alpha \rho_k m_k f_k^3 \\ s.t. \quad & c_k m_k f_k \geq \frac{1}{\tau \xi^{\frac{1}{v}}} \\ & m_k \in \{1, 2, ..., M\} \\ & f_k \in \mathcal{F} \end{aligned} \tag{11}$$

where M is the total number of servers in the cluster, and $\mathcal{F} = \{f^{(1)}, f^{(2)}, \ldots, f^{(l)}\}$ is the CPU frequency set with l as the number of the available CPU frequencies.

B. Multiple VM Servers on One Physical machine

Suppose each physical machine in the cluster is configured as a host hosting multiple homogeneous VM servers. We propose two ways to decide the number of the turn-on physical machines and the number of active VM servers.

1) Capacity Allocation with Pattern Identification: When the workload in the server cluster, i.e. the arrival rate of the requests, follows a pattern during one day or one week, we can utilize this pattern to turn on a fixed number of physical machines m_0 in certain period according to the peak workload during this period. Then, we fine tune the number of active VM servers on the m_0 machines to handle the workload fluctuations in this period. In this way, the expected power consumption of cluster, according to Eq.(5), is $\mathbb{E}(P^t) = m_0(P^I + \beta_0 \rho_k f_k^3) + \beta_1 \rho_k g_k f_k^3$, where g_k is the total number of active VM servers on the m_0 machines at the kth sampling period. Note that the g_k active servers are arranged on the m_0 machines in a round-robin way, with each machine can be assigned $\left\lfloor \frac{g_k}{m_0} \right\rfloor$ or $\left\lceil \frac{g_k}{m_0} \right\rceil$ VM servers, where $\lfloor z \rfloor$ and $\lceil z \rceil$ denotes the nearest integer smaller than z or larger than z, respectively. The SLA constraint is still in the form of Eq.(10) but with m replaced by g_k. In this situation, our goal can be formulated as the following problem,

$$\begin{aligned} \min_{g_k, f_k} \; & m_0(P^I + \beta_0 \rho_k f_k^3) + \beta_1 \rho_k g_k f_k^3 \\ s.t. \quad & c_k g_k f_k \geq \frac{1}{\tau \xi^{\frac{1}{v}}} \\ & g_k \in \{1, 2, ..., M n_0\}, \\ & f_k \in \mathcal{F}, \end{aligned} \tag{12}$$

where n_0 is the number of VMs on each physical machine, and Mn_0 is the total number of VMs in the server clusters. Note that if g_k obtained from Problem (12) is larger than $m_0 n_0$, workload burst may happen in the kth sampling period. When the workload bursts and does not follow the normal patterns, we need to add physical machine to support the $(g_k - m_0 n_0)$ active VM servers.

2) Capacity Allocation without Pattern Identification: When the workload in the server cluster is highly dynamic or cannot be predicted beforehand, pattern-based capacity allocation cannot be leveraged. Suppose g_k VM servers should be activated at the kth sampling period, and the number of physical machine that should be turn on is $m_k = \left\lceil \frac{g_k}{n_0} \right\rceil$ with n_0 as the number of VM servers hosted by each machine. Thus the expected power consumption in the server, according to Eq.(5), is $\mathbb{E}(P^t) = \left\lceil \frac{g_k}{n_0} \right\rceil (P^I + \beta_0 \rho_k f_k^3) + \beta_1 \rho_k g_k f_k^3$. In this situation, our goal can be formulated as the following problem,

$$\begin{aligned} \min_{g_k, f_k} \; & \left\lceil \frac{g_k}{n_0} \right\rceil (P^I + \beta_0 \rho_k f_k^3) + \beta_1 \rho_k g_k f_k^3 \\ s.t. \quad & c_k g_k f_k \geq \frac{1}{\tau \xi^{\frac{1}{v}}} \\ & g_k \in \{1, 2, ..., M n_0\} \\ & f_k \in \mathcal{F} \end{aligned} \tag{13}$$

IV. Overview of TailCon Framework

We design an online computing capacity allocation scheme, referred to as TailCon, to realize tail probability control and power minimization in the server cluster even under the dynamic workload. Since the workload in modern server clusters are complex and highly dynamic, we integrate multiple strategies to provide the desired performance, including the patter-based capacity planning, resource pre-allocation according to the workload change point detection and the optimal capacity provision based on the online analysis to the response time distribution.

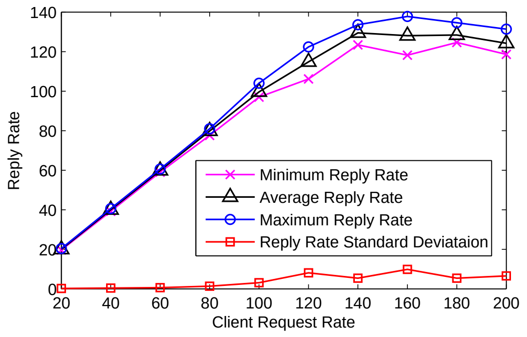
A. pattern-based capacity planning

The workload in some server cluster follows a pattern which can be summarized from the historical record of the client requests in a long period. In this case, the workload in the server cluster is highly predictable. Take 24-hour HTTP trace [8] in Fig 7(a) as an example, we can see that the HTTP request reaches its peak arrival rate during 2:00 to 10:00, and stays under the a threshold 80req/s in the period from 0:00 to 2:00 and from 10:00 to 24:00. If the same pattern can be recognized in everyday's HTTP request trace, we can apply pattern-based capacity planning as illustrated in Section III-B1. The pattern based capacity planning needs to know the capacity of each server at the design time. Take a single Apache server in our implementation as an example, we use a server performance benchmarking tool AUTOBENCH [18] to generate the HTTP requests and test the maximum concurrent requests that our server can support, where each client request requires a dynamic PhP webpage with the reply size X following Pareto distribution ($\nu = 1.2$ and $\hat{x} = 1$KB). The reply rate versus the arrival rate of the requests is in Fig. 6(a), which shows that the server saturates at a concurrency of 120req/s. The corresponding response time is shown in Fig. 6, where the mean response time is around 500ms when the server begins to saturate. Based on the capacity of each server, according to the patterns of the request arrival rate and their reply sizes, we can decide how many servers to activate. For example, if the pattern in the 24-hour HTTP trace (Fig. 7(a)) can be predicted one day before and the reply sizes follows a known distribution like the Pareto distribution with predictable parameters (i.e. $\nu = 1.2$ and $\hat{x} = 10$KB), we can provide 2 servers during 2:00 to 10:00 to serve the requests with arrival rate between 80req/s and 230req/s, and we can allocate 1 server in the other periods during the day to serve the requests with arrival rate under 80req/s. In each period, we can fine tune the provision of computing capacity by adjusting the CPU frequencies and the number of activated servers to adapt to the workload variation.

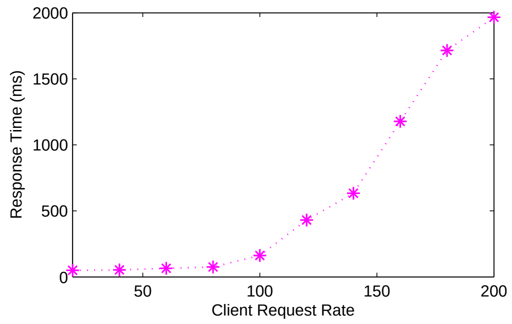
The pattern-based capacity planning is only applicable to clusters with predictable workload following observable patterns. The pattern-based capacity planning facilitates the structural management to the cluster resources and reduces the hardware tearing since frequently turning on/off the physical machines can be avoided.

B. Capacity Pre-Allocation based on Workload Change Point Detection

The workload in the cluster may change at the beginning of a sampling period, which may degrade the service to the client if the capacity provision is adjusted at the end of the sampling period. In this case, we use change point detection to detect the workload changes at the beginning of a sampling period and take actions accordingly. To be specific, the frontend analyzes the request logs online. For the first 1000 client requests during a sampling period that contains replies to 10000 requests, the frontend checks whether more than 1000ξ requests's response time exceeds the threshold τ. If this situation happens, the workload should experience a sudden burst that the current capacity cannot serve, therefore, if each machine is configured as a single server, we adjust the number of active servers (m_k) to be $m_k = \left\lceil \frac{d}{1000\xi} m_k \right\rceil$; if each server is configured to host multiple VM servers, we adjust the number of active VM servers (g_k) to be $g_k = \left\lceil \frac{d}{1000\xi} g_k \right\rceil$, where d is the number of requests with response time larger than τ among the first 1000 requests. The machines that hold the newly activated servers or VM servers will run at the same CPU frequencies with the machines already turned on.



(a) Reply Rate (req/s)



(b) Response Time (ms)

Fig. 6: Concurrency Test for a Single Server under Different Request Arrival Rate

C. Optimal Capacity Allocation based on Response Time Feedback

For every sampling period, TailCon scheme computes the optimal computing capacity. If each physical machine in the cluster is configured as a single server, the optimal computing capacity is derived by solving the problem in (11). If each physical machine in the cluster is configured as a host for several VM servers, the optimal computing capacity is derived by solving the problem in (12) or (13) depending on whether the pattern-based capacity planning is applied. To solve these optimization problems, we need to estimate the coefficient c_k and get the CPU utilization of each server ρ_k. Moreover, we need to analyze the request response time distribution online to achieve the distribution parameters ν, $\hat{y}$.

1) Online Analysis to Response Time Distribution: We set the sampling period for online capacity allocation as the period long enough to finish serving 10000 client requests. We choose 10000 replies to the client requests as the sampling period to avoid frequent adjustment to the computing capacity allocation and to ensure enough samples of the request response time can be collected in each sampling period. For every 10000 requests served by the cluster, the frontend extracts 1000 samples uniformly in the 10000 requests to analyze their response time distribution and find the Pareto distribution parameter ν, $\hat{y}_k$. This can be achieved by applying the power-law fitting tool [17] implemented in python scripts. For the response time distribution analysis, we use 1000 samples instead of all the 10000 requests to reduce the real-time computation cost.

2) Estimation of c_k based on Recursive Least Square Method: Since the workload variation can influence the relation between the response time distribution parameter $\hat{y}$ and the provision of the computing resources m_k (or g_k) and f_k, we need to estimate c_k online using Recursive Least Square (RLS) method with selective forgetting factor. Since c is the coefficient in Eq.(9), at the end of the kth sampling period, we compute c by solving a weighted least squares problem,

$$\min_{c} \sum_{i=1}^{k} w(k,i)\big(\frac{1}{\hat{y}_i} - cm_i f_i\big)^2, \tag{14}$$

where $w(k,i)$ is the weight factor on the square error $\big(\frac{1}{\hat{y}_i} - cm_i f_i\big)^2$ for $i = 1, \ldots, k$,

$$w(k,i) = \begin{cases} 1, i = k, \\ \prod_{j=i+1}^{k} \gamma_j, i < k. \end{cases} \tag{15}$$

In Eq.(15), γ_j is a forgetting factor adopted at the jth sampling period ($j = 1, \ldots, k$), $0 \leq \gamma_j \leq 1$. The solution to problem (14) is a correct estimation of c in the kth sampling period but will be used as an estimation of c in the $(k+1)$th sampling period, denoted as $\hat{c}_{k+1}$. $\hat{c}_{k+1}$ replaces c_k in the optimization problem (11), (12) and (13) to deduce m_{k+1} and f_{k+1}, which will be applied as the optimal provision of the computing capacity to serve the requests in the $(k+1)$th sampling period. Therefore, at the kth sampling period, when the measured $\hat{y}_k$ deviates from its estimated value $1/(\hat{c}_k m_k f_k)$ (Eq. (9)), we consider the workload changes abruptly and set $\gamma_k = 0$ to eliminate the influence of the historical measurement $\hat{y}_{k-1}, \ldots, \hat{y}_1$ in estimating c. Otherwise, we will set $\gamma_k = 1$ to make an accurate estimation of c by leveraging the historical data.

$$\gamma_k = \begin{cases} 0, & |\hat{y}_k \hat{c}_k m_k f_k - 1| \leq \delta, \\ 1, & |\hat{y}_k \hat{c}_k m_k f_k - 1| > \delta, \end{cases} \tag{16}$$

where $\hat{y}_k \hat{c}_k m_k f_k$ is the ratio of $\hat{y}_k$ to its estimate $1/(\hat{c}_k m_k f_k)$, and we set δ as threshold (a small positive number) to judge whether the workload changes in the kth sampling period. A ratio $\hat{y}_k \hat{c}_k m_k f_k$ close to 1 indicates no workload variation during the kth sampling period and vice versa.

The least squares problem in Eq.(14) can be solved in a recursive way, and it is easy to show the solution is

$$\hat{c}_{k+1} = \frac{h_k}{b_k}, \tag{17}$$

where

$$\begin{aligned} h_k &= \gamma_k h_{k-1} + \frac{1}{\hat{y}_k} m_k f_k, \\ b_k &= \gamma_k d_{k-1} + (m_k f_k)^2. \end{aligned} \tag{18}$$

Since we always use $\hat{c}_{k+1}$ computed at the kth sampling period as a predict to c in the $(k+1)$th sampling period, we inexplicitly assume the workload does not change in the two sampling periods. If the workload changed abruptly at the $(k+1)$th sampling period, it can only be detected at the end of the $(k+1)$th sampling period. It is known as one-sampling-period delay in the feedback-based capacity planning schemes. This is also the reason that we introduce capacity pre-allocation strategy based on workload change point detection as a compensation to the one-sampling-period delay, as illustrated in Section IV-B.

3) Solution to Optimization Problem: At the end of each sampling period, we need to derive the optimal provision of the computing capacity for the next sampling period. For example, m_k (or g_k) and f_k are derived at the end of the $(k-1)$th sampling period. Take finding m_k and f_k by solving problem (11) as a general case. At the end of the $(k-1)$th sampling period, the parameters for the response time distribution like ν, $\hat{y}$ are achieved by applying power-law fitting tool [17], and $\hat{c}_k$ is derived according to Eq.(18). Meanwhile, the frontend measures the CPU utilization of each active server and uses it as ρ_k in the optimization problem (11). To derive the optimal m_k and f_k, the strategy is summarized below.

- Step1: for each CPU frequency $f^{(i)}$ in the available CPU frequency set $\mathcal{F}$, we compute the minimum number of active servers $m^{(i)}$ that can satisfy the SLA constraint, $m^{(i)} = \left\lceil 1/(\tau\xi^{\frac{1}{\upsilon}} \hat{c}_k f^{(i)}) \right\rceil$. And $m^{(i)}$ will be rounded to the maximum number of servers in the cluster M if its value exceeds M.
- Step2: for each pair $f^{(i)}, m^{(i)}$, we will use it to compute the expected power consumption of cluster $\mathbb{E}(P^t)$, and we choose the pair $f^{(i)}, m^{(i)}$ that can result in the minimum expected power consumption as the optimal result f_k, m_k.

The above strategies can also be applied to solve optimization problem (12) and (13) with $m^{(i)}$ replied by $g^{(i)}$. Since DVFS-enabled machine can only provide limited number of available CPU frequencies, i.e. usually less than 12 available CPU frequencies, enumerating the CPU frequencies takes $O(l)$ computing cost with l as the number of the values in $\mathcal{F}$.

D. TailCon Structure

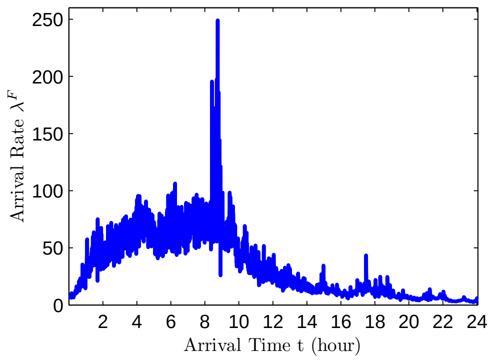
Tailcon is implemented as executable scripts at the frontend. The frontend remotely turns on/off the backend servers and uses the dynamic load-balancing strategy to dispatch the incoming requests to the active servers. The frontend uses script to process the request log file like the Apache piped logs online. During a sampling period, the frontend analyzes the first 1000 request replies to see whether the capacity pre-allocation is needed. After finish serving 10000 sampling periods, the frontend calls the power-law fitting tool written in Python to get the Pareto distribution parameters for the request response time. Then, the frontend computes $\hat{c}$ by applying Eq.(18) and find f_k, m_k or f_k, g_k by solving the proper optimization problem in (11), (12) and (13). Finally, the frontend adjusts the number of active servers or VM servers and send the CPU frequency scaling command to the physical machines that hold these active servers. This procedure repeats in every sampling period till the end of the runtime.

V. Experiment Setup

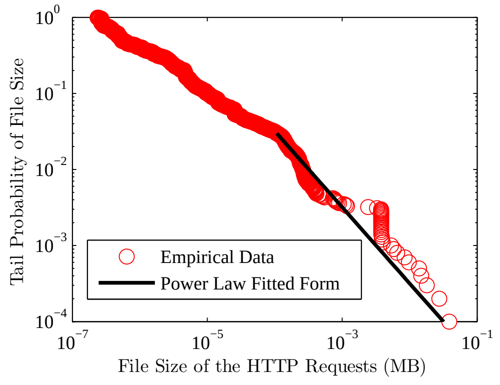
To test the effectiveness of TailCon scheme in different server cluster configurations, we implement both experiments and emulation based on real-world HTTP traces [8]. And we compare our result to a static computing capacity planning scheme PowerFix.

A. Emulation Setup

Due to the experimental resource limitation, we cannot conduct experiments on a large-scale server cluster with each physical machine configured as a single server, hence we build a discrete event emulator based on J-Sim [19] to simulate a server cluster with totally 15 servers. The frontend decides how many servers to turn on and which CPU frequency for each server to run in each sampling period. The frontend dispatches the customer requests to the active servers in a



(a) Arrival Rate of Requests



(b) Power-Law Fitting of File Size

Fig. 7: 24-Hour HTTP Trace Information

round-robin policy. The server uses the processor sharing discipline to schedule the incoming requests. We implement TailCon scheme as a function in the frontend.

We use the inter arrival time of a real world 24-Hour HTTP trace [8] to generate the customer requests, where the arrival rate is shown in Fig. 7(a). We can see that the customer arrival rate λ^F is highly dynamic with the maximum and minimum values as 256 and 2.4 respectively. Since the server serves incoming requests using processor sharing discipline, the required service time of each request is proportional to the size of the webpage required by the request [11]. Therefore, we configured our emulation so that each incoming request requires a service time that follows a Pareto distribution. We apply the power law fitting tool [17] to the file size of the HTTP requests to derive the Pareto index ν, where $\nu = 1.3$. Fig. 7(b) shows the power-law distribution fitting of 10-minutes traces from 17:26 to 17:36 in the HTTP trace [8]. In our emulation, we set $\tau = 1$s and $\xi = 1\%$ as SLA constraint, indicating 99% of the requests should receive its reply within 1 second.

The emulator simulates the power consumption in the server cluster using the power model derived based on our implementation in Section II and is summarized in Table II. The power consumption of the server cluster is computed by the frontend according to Eq.(4).

B. Experiment Setup

To test the effectiveness of TailCon scheme in the server cluster with each machine supporting several VM servers, we implement 2 homogeneous physical machines with totally 6

TABLE II: Parameters Setup in Emulation and Experiment

Parameter	Value
$\mathcal{F}$ (GHz)	(2.528, 2.527, 2.394, 2.261, 2.128, 1.995, 1.862, 1.729, 1.463, 1.33, 1.197)
P^b with single server on single machine (watts for $f^{(i)} \in \mathcal{F}$)	(97, 90, 85.5, 82.2, 78.1, 75.4, 72.3, 69.6, 65.2, 64, 61.4)
P^I (watts) in Eq.(4)	58.3648
α (watts/GHz^3) in Eq.(4)	2.1143
P^b with 0 active VM server on single machine (watts for $f^{(i)} \in \mathcal{F}$)	(67.9, 64.8, 64.3, 64.8, 63.3, 63.2, 62.7, 62.4, 62.2, 62.1, 61.4)
P^b with 1 active VM server on single machine (watts for $f^{(i)} \in \mathcal{F}$)	(76.1, 67.5, 66, 64.5, 64.4, 63.8,63.6, 63.5, 62.9, 62.7, 62.4)
P^b with 2 active VM servers on single machine (watts for $f^{(i)} \in \mathcal{F}$)	(84.5, 75.9, 73.2, 71.6, 67.5, 67, 65.4, 63.6, 63.4, 63.4, 63.3)
P^b with 3 active VM servers on single machine (watts for $f^{(i)} \in \mathcal{F}$)	(92.6, 83.9, 80.5, 76.5, 73.8, 70.7, 66.2, 66.8, 64.4, 63.2, 63.4)
P^I (watts) in Eq.(5)	59.6221
β_0, β_1 (watts/GHz^3) in Eq.(5)	(0.2078, 0.4892)
Parameter in Distribution of x	$\nu = 1.3$, $\hat{x} = 1$KB

VM servers. We use VirtualBox as the virtualization tool. Each physical machine (2 Quad-core Intel Xeon processors and 4 GB RAM running Centos 6.2 operating system) holds 3 homogeneous VM servers with each VM server implemented as an Apache server serving dynamic PhP webpages whose sizes follow Pareto distribution. Each physical machine is DVFS-enabled, and the available CPU frequencies are shown in Table II. The power consumption of the physical machine follows the power consumption model in Eq.(5) with the parameters summarized in Table II. The frontend is located on an independent machine (2 Intel dual-core processors and 4 GB RAM) and is implemented as an Apache server with module mod_proxy_balancer enabled for load-balancing management. The client is located on another machine (Intel Dual-core processor and 3 GB RAM) running Linux operating system. We leverage server performance benchmarking tool Httperf [20] to generate the client requests in around 50 minutes to test the effectiveness of the proposed scheme, and the arrival rate of the client requests during the 50 minutes runtime is shown in Fig. 8. The client requests are requiring the PhP webpages whose sizes X follow Pareto distribution with $\nu = 1.3$ and $x = 1$KB. TailCon scheme is implemented as executable Bash scripts at the frontend, which works according to the principles introduced in Section IV-D. The power consumption of the physical machines is measured by power meter. Since the client requests during the runtime is highly dynamic and is unpredictable, hence we did not implement the pattern-based capacity planning. Therefore, TailCon scheme implemented in our experiment solves problem (13) in each sampling period for optimal capacity allocation.

C. Baseline

We use PowerFix as a baseline capacity allocation scheme. PowerFix turns on a fixed number of servers (or VM servers) and let the turn-on machines run at a fixed CPU frequency during the overall runtime. The m (or g) and f in PowerFix are computed by minimizing the power consumption at the peak arrival request rate. In our emulation based on the 24-hour HTTP trace [8] (see Fig. 7(a)), we configure PowerFix by turning on 15 servers running at the fixed CPU frequency

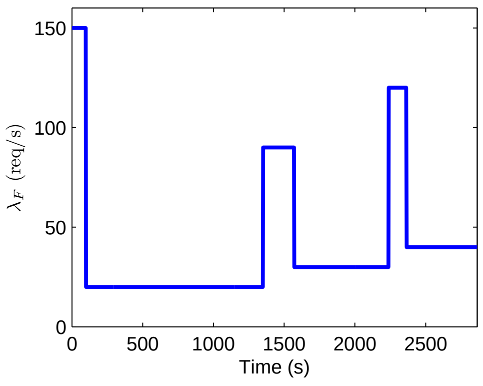


Fig. 8: HTTP Trace in Our Experiment

2.528GHz during the runtime. And in our experiments based on the requests arrival trace (Fig. 8), we configure PowerFix by activating 6 VMs on the 2 machines with each machine running at 2.394GHz.

VI. Performance Evaluation

We conduct emulation and experiment using the setup in Section V to evaluate the performances of both TailCon scheme and PowerFix scheme for controlling the tail percentile of request response time stay under 1%, where we set τ, the response time upper bound, as 1s.

A. Emulation Results: Single Server on Single Machine

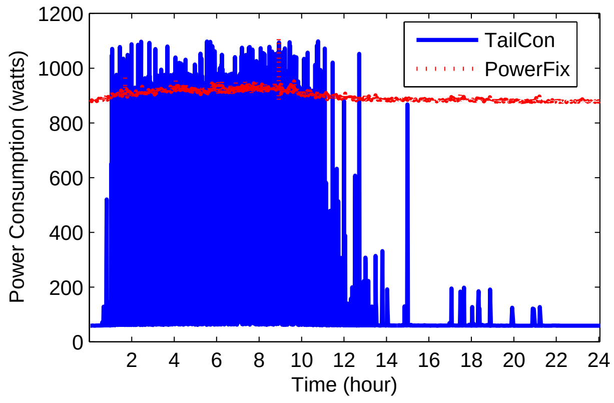
We simulate a server cluster serving 24 hour HTTP requests with arrival rate in Fig. 7(a) in the emulation. The power consumption of the cluster under TailCon scheme and PowerFix scheme is shown in Fig. 9(a). Since TailCon provides the optimal computing capacity in adaptation to the workload variation, the power consumption shows a similar variation trend with the HTTP request arrival rate during the 24 hours. In contrast, PowerFix provides the computing capacity based on the peak workload, it results in higher power consumption than TailCon during most of the runtime. The request response time for the 24 hour HTTP traces under TailCon scheme and PowerFix scheme is shown in Fig. 9(b). As we can see, there are around $270,0000$ requests in the runtime, and most of the requests stay under 1s in both TailCon and PowerFix.

B. Experiment Result: Multiple VM Servers on Each Machine

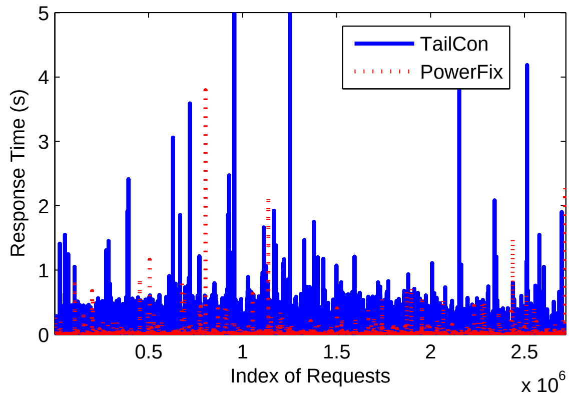
We conduct experiments to test 2 physical machine with totally 6 VM servers serving HTTP requests with arrival rate in Fig. 8. The power consumption of the cluster under TailCon scheme and PowerFix scheme is shown in Fig. 10(a). We can see the power consumption under TailCon is smaller than the power consumption under PowerFix due to the dynamic capacity allocation. The response time for the HTTP requests in the experiment is shown in Fig. 10(b). As we can see, there are around $11,0000$ requests in the runtime, and most of the requests stay under 1s in both TailCon and PowerFix.

C. Performance Comparison

We summarize the performance results of TailCon and PowerFix in Table III. For both the emulation emulating the large-scale server cluster based on real-world HTTP traces and the experiment testing the small-scale server cluster with each machine configured to support multiple VM servers, TailCon can save the power consumption in comparison with the static



(a) Power Consumption



(b) Response Time

Fig. 9: Performance of TailCon and PowerFix in the Emulation based on 24-Hour HTTP Trace

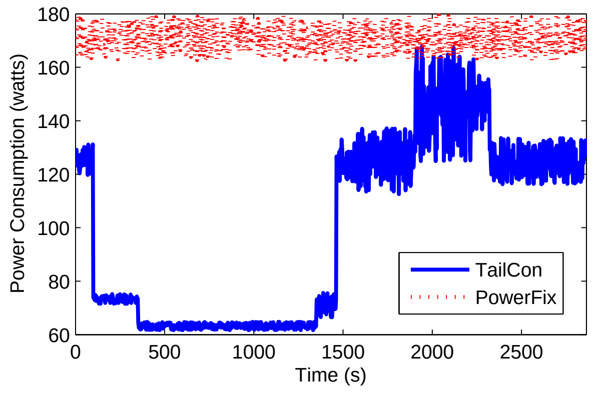
TABLE III: Performance Comparison of Two Schemes

Performance	TailCon	PowerFix
Average Power Consumption (Emulation)	362.0530 watts	899.8134 watts
Average Power Consumption (Experiment)	100.1276 watts	170.9093 watts
Response Time Tail Percentile (Emulation)	1.6546e-05	1.8384e-06
Response Time Tail Percentile (Experiment)	3.4055e-04	1.7464e-05

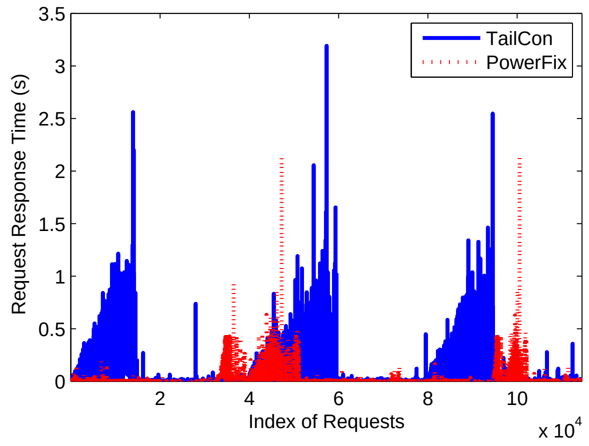
capacity provision scheme and at the same time keep the response time percentile stay under the setpoint.

VII. Related Work

A significant amount of work focuses on power-aware computing capacity planning in server cluster [10], [11], [21]–[25]. In these works, the SLAs are expressed with either the mean value of the response time in server systems [10], [21], [22], or based on the assumption on the limited computing resource in the server clusters [23]–[25]. With the growing scale of the modern server clusters, the overall computing capacity could be much higher than the customer demand even under the worst workload situation. The power consumption and operating cost supersedes the computing resource and becomes the major concern. To meet the SLA represented by the percentile of response time guarantees, the previous works listed above cannot work. Closely related to this paper, Bertini et al. [26] used deadline missing probability as



(a) Power Consumption



(b) Response Time

Fig. 10: Performance of TailCon and PowerFix in the Experiment with Dynamic HTTP Requests

the performance metric in web server clusters and proposed an online feedback control scheme to control the Quality of Service (QoS). However, they did not take the power management into consideration. Wang et al. [9] proposed an computing capacity planning strategy to enforce the response time percentile constraint while minimizing power, but it is an offline strategy that cannot be applied for real-time operation of the server cluster. In contrast, we propose TailCon scheme that can dynamically provide the computing capacity online according to the varying workload and can minimize the power consumption subject to the SLA constraint.

VIII. Conclusion

To provide SLA guarantees and minimize the power consumption in the server cluster, we propose TailCon scheme to provide the optimal computing capacity in the server cluster in adaptation to the workload variation. We consider different server cluster configuration in TailCon scheme design, and integrate multiple capacity strategies in TailCon scheme to provide good capacity planning results. We implement emulation based on real-world HTTP trace and experiments to test the performance of TailCon, and the results demonstrate the effectiveness of TailCon to control the tail probability of the response time and at the same time reduce the power consumption in comparison with a baseline scheme PowerFix.

References

[1] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger, “Oceano-SLA Based Management of a Computing Utility,” in *in Proceedings of the IEEE/IFIP International Symposium on Integrated Network Management*, 2001, pp. 855–868.

[2] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, “No Power Struggles: Coordinated Multi-Level Power Management for the Data Center,” *ACM SIGPLAN Notices*, vol. 43, no. 3, pp. 48–59, 2008.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon’s Highly Available Key-Value Store,” *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205–220, 2007.

[4] M. Arlitt and C. Williamson, “Internet Web Servers: Workload Characterization and Performance Implications,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 631–645, 2002.

[5] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, “Changes in Web Client Access Patterns: Characteristics and Caching Implications,” *World Wide Web*, vol. 2, no. 1, pp. 15–28, 1999.

[6] P. Barford and M. Crovella, “A Performance Evaluation of Hyper Text Transfer Protocols,” *in Proceedings of the International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS’99)*, 1999, pp. 188–197.

[7] M. Crovella and A. Bestavros, “Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835–846, 2002.

[8] “Web Caching Project”, “trace:sd.sanitized-access.20110726,” http://www.ircache.net, July 26th 2007.

[9] S. Wang, J. Liu, X. Liu, and J. Chen, “Power-Saving Design for Server Farms with Response Time Percentile Guarantees,” *in Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’12)*, 2012, pp. 273–284.

[10] M. Lin, A. Wierman, L. Andrew, and E. Thereska, “Dynamic Right-Sizing for Power-Proportional Data Centers,” *in Proceedings of IEEE INFOCOM,*, 2011, pp. 1098–1106.

[11] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, “Managing Server Energy and Operational Costs in Hosting Centers,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, pp. 303–314, 2005.

[12] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, “Optimal Power Allocation in Server Farms,” *in Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems*, 2009, pp. 157–168.

[13] W. Leland and T. Ott, “Load-Balancing Heuristics and Process Behavior,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 14, no. 1, pp. 54–69, 1986.

[14] M. Harchol-Balter and A. Downey, “Exploiting Process Lifetime Distributions for Dynamic Load Balancing,” *ACM Transactions on Computer Systems (TOCS)*, vol. 15, no. 3, pp. 253–285, 1997.

[15] P. Barford and M. Crovella, “Generating Representative Web Workloads for Network and Server Performance Evaluation,” in *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 1, pp. 151–160, 1998.

[16] D. Chotikapanich, “"Chapter 7: Pareto and Generalized Pareto Distributions", Modeling Income Distributions and Lorenz Curves,” p. 121ĺC122.

[17] A. Clauset, C. Shalizi, and M. Newman, “Power-Law Distributions in Empirical Data,” *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009.

[18] J.T.Midgley, “Autobench,” http://xenoclast.org/autobench.

[19] H. Tyan, A. Sobeih, and J. Hou, “Design, Realization and Evaluation of a Component-Based, Compositional Network Simulation Environment,” *Simulation*, vol. 85, no. 3, p. 159, 2009.

[20] D. Mosberger and T. Jin, “httperfąła Tool for Measuring Web Server Performance,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[21] D. Meisner, B. Gold, and T. Wenisch, “PowerNap: Eliminating Server Idle Power,” *in Proceeding of the 14th international conference on Architectural support for programming languages and operating systems (ASPLOS ’09)*, 2009, pp. 205–216.

[22] S. Wang, J. Chen, J. Liu, and X. Liu, “Power Saving Design for Servers under Response Time Constraint,” *in Proceedings of 22th Euromicro Conference on Real-Time Systems, (ECRTS’10)*, 2010, pp. 123–132.

[23] A. Chandra, P. Goyal, and P. Shenoy, “Quantifying the Benefits of Resource Multiplexing in On-Demand Data Centers,” *in Proceedings of the First Workshop on Algorithms and Architectures for Self-Managing Systems*, 2003.

[24] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, “Qos-Driven Server Migration for Internet Data Centers,” *in Proceedings of International Workshop on QoS Quality of Service*, 2002, pp. 3–12.

[25] B. Urgaonkar and P. Shenoy, “Cataclysm: Handling Extreme Overloads in Internet Services,” in *in Proceedings of ACM Principles of Distributed Computing (PODC’04), St. Johnqŕs, Newfoundland, Canada*, 2004.

[26] L. Bertini, J. Leite, and D. Mossé, “Statistical QoS Quarantee and Energy-Efficiency in Web Server Clusters,” *in Proceedings of 19th Euromicro Conference on Real-Time Systems, (ECRTS’07)*, 2007, pp. 83–92.