RESEARCH-ARTICLE

PREACT: Predictive Resource Allocation for Bursty Workloads in a Co-located Data Center

DINGYU YANG, Zhejiang University, Hangzhou, Zhejiang, China

ZIYANG XIAO, Zhejiang University, Hangzhou, Zhejiang, China

DONGXIANG ZHANG, Zhejiang University, Hangzhou, Zhejiang, China

SHUHAO ZHANG, Nanyang Technological University, Singapore City, Singapore

JIAN CAO, Shanghai Jiao Tong University, Shanghai, China

GANG CHEN, Zhejiang University, Hangzhou, Zhejiang, China

Published: 12 August 2024

ICPP '24: the 53rd International Conference on Parallel Processing
August 12 - 15, 2024
Gotland, Sweden

ICPP '24: Proceedings of the 53rd International Conference on Parallel Processing (August 2024)
https://doi.org/10.1145/3673038.3673135
ISBN: 9798400717932

PREACT: Predictive Resource Allocation for Bursty Workloads in a Co-located Data Center

Dingyu Yang
The State Key Laboratory of Blockchain and Data Security, Zhejiang University & Alibaba Group
Hangzhou, China

Ziyang Xiao
Zhejiang University
Hangzhou, China
xiaoziyang@zju.edu.cn

Dongxiang Zhang*
Zhejiang University
Hangzhou, China
zhangdongxiang@zju.edu.cn

Shuhao Zhang
Nanyang Technological University
Singapore
shuhao.zhang@ntu.edu.sg

Jian Cao
Shanghai Jiao Tong University
Shanghai, China
cao-jian@cs.sjtu.edu.cn

Gang Chen
Zhejiang University
Hangzhou, China
cg@zju.edu.cn

ABSTRACT

Co-locating online latency-critical (LC) services with best-effort (BE) batch jobs in the same server has been widely adopted by modern data centers to improve resource utilization. Various approaches have been proposed to maximize the resources allocated to the BE jobs without SLO (service level objective) violation. However, when facing bursty workloads, existing solutions suffer from poor performance because they cannot react promptly to the sudden and sharp increase of LC service requests. Consequently, these methods result in either a high violation rate of the SLO constraint or low resource utilization caused by conservative allocation strategies.

In this paper, we propose PREACT as a predictive and agile resource allocation manager to support bursty workloads in a co-located data center. We devise an accurate and lightweight predictor based on a decomposable time series model to estimate the QPS (queries per second) for LC services in the next time window. Given the predicted QPS, we propose an SLO profiling model based on queuing theory and optimize it with multilayer perceptrons. The model is able to determine the maximum amount of resources that can be allocated to the BE jobs without any SLO violation. We conduct extensive experiments using real trace logs of multiple LC services with bursty workload patterns in a major E-commerce promotion campaign in 2021. The results establish the superiority of PREACT when handling bursty workloads — it incurs the lowest SLO violation and achieves comparable or higher CPU utilization than prior resource managers in a co-located data center.

CCS CONCEPTS

• **Applied computing** → **Data centers**; *Enterprise resource planning*; Enterprise computing infrastructures.

KEYWORDS

Bursty Workloads, Data Center, Predictive Resource Allocation

ACM Reference Format:
Dingyu Yang, Ziyang Xiao, Dongxiang Zhang, Shuhao Zhang, Jian Cao, and Gang Chen. 2024. PREACT: Predictive Resource Allocation for Bursty Workloads in a Co-located Data Center. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3673038.3673135

*Corresponding author

ICPP '24, August 12–15, 2024, Gotland, Sweden

ACM ISBN 979-8-4007-1793-2/24/08
https://doi.org/10.1145/3673038.3673135

1 INTRODUCTION

In a modern data center with massive-scale computing infrastructure, increasing a small percentage of resource utilization can lead to a considerable reduction in terms of operating expenses and carbon emissions. It has become a popular practice for giant cloud service providers such as Google [31], Alibaba [4], Microsoft [16], and Facebook [13] to co-locate the latency-critical (LC) services with the best-effort (BE) batch jobs in the same server. In such a co-located environment, LC services normally have higher priority than BE jobs, which are treated as "second-class citizens" and allowed to be deferred or restarted to release resources for LC services.

There have been numerous resource allocation strategies proposed with the objective of ensuring the SLO (service level objective) requirement for the LC services, and in the meanwhile allocating as many resources as possible to the BE jobs. For example, feedback-based resource managers such as Bistro [13], Heracles [22], PARTIES [3] and Perfiso [16] continuously monitor the real-time resource usage at runtime and determine the amount of idle resources that can be allocated to the BE jobs. Alternatively, predictive resource managers such as Scavenger [17] and Autopilot [26] leverage moving-window based statistics to dynamically adjust the resource quota for the BE jobs. However, when facing bursty workloads, these approaches cannot react promptly to the abrupt and dramatic increase pattern of online requests. They fail to pre-allocate sufficient resources for LC services and suffer from performance degradation.

Figure 1 illustrates a real application scenario with bursty workloads from our major E-commerce promotion campaign, and we plot the QPS (query per second) of two typical LC services related to online shopping. For reasons of commercial confidentiality, the values of QPS are normalized into $[0, 1]$. Compared with daily workload (plotted in orange curves), there is a sudden and sharp

increase pattern, which quickly boosts the traffic to a peak with 5-10 times higher volume. Then, the workload of these two LC services gradually falls to the normal level.

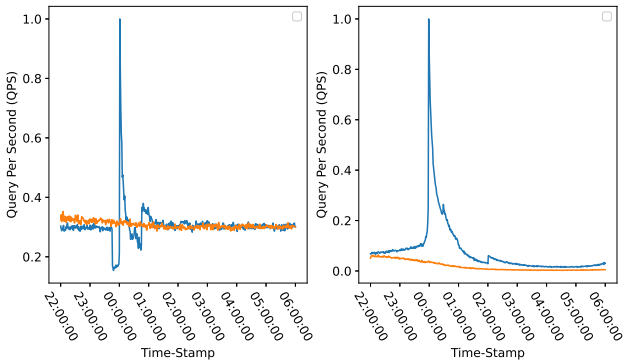


Figure 1: Comparison of daily QPS with bursty workloads for two LC services in a major E-commerce promotion campaign in 2021. The QPS values are normalized in [0,1] due to commercial confidentiality.

The scenario in figure 1 is particularly challenging to tackle as it requires the resource manager to promptly release the resources allocated to the BE jobs to avoid SLO violation. When the workload starts to decline, ideally, we need to gradually increase the quota for the BE jobs to improve resource utilization. However, in practice, a data center may choose a conservative strategy to avoid SLO violations. For example, our data center currently downgrades the priority of BE jobs, by preventing their job submissions, deferring their job scheduling, or even killing some resource-intensive jobs to satisfy the requirements of LC services. This process may last for several hours, which are painful for internal users as they are unable to submit analytical tasks.

In this paper, we propose PREACT as a predictive and agile resource allocation manager to better support bursty workloads. PREACT is associated with a lightweight predictor to accurately estimate the QPS of an LC service in the t-th sliding time window, denoted by $QPS(t)$. To implement the predictor, we adopt the idea of a decomposable time series model and decompose $QPS(t)$ into a trend signal $g(t)$ and a bursty event signal $e(t)$. Then, we devise two separate regression models to estimate $g(t)$ and $e(t)$, respectively. Compared with deep learning-based models for time series forecasting, our model only requires observations in recent time windows for model fitting and is more agile and accurate when predicting bursty workloads. With the predicted QPS, we construct an SLO profiling model to capture the relationship between the response time of an LC service and its arrival rate, under a specified resource constraint. The SLO profile allows us to find the minimum resources that can be allocated to LC services without violating the SLO constraint. Since we can allocate all the remaining resources to the BE jobs, this is equivalent to finding the resource quota for the BE jobs. To learn the SLO profile, we build a parametric model based on queuing theory and optimize it with multilayer perceptrons.

Compared with prior works, we highlight that this paper intends to tackle the bursty workload scenario with real LC services in a private production cloud, where the application-level information of LC services is available. To the best of our knowledge, this is the first work to quantitatively profile SLO with CPU resources and support an adjustable response time (RT) threshold for an LC service.

To validate the effectiveness of our approach in real-world scenarios, we construct an environment using real trace logs collected from multiple LC services in a major E-commerce promotion campaign in 2021. Extensive experiments are conducted to verify the effectiveness of PREACT in terms of supporting bursty workloads. The results show that PREACT achieves the lowest SLO violation rate, and in the meanwhile exhibits comparable or even higher resource utilization rates than existing resource allocation managers for co-located workloads. To validate the superiority, we perform a comprehensive analysis to reveal the limitations of comparison approaches when handling bursty workloads and explain the reasons why PREACT can overcome these challenges.

2 PRELIMINARIES AND PROBLEM DEFINITION

In this section, we present the preliminaries, frame the research scope, and formulate the resource allocation problems.

2.1 Co-location in a Data center

In this paper, we study the problem of resource allocation in a data center with Kubernetes clusters. In each machine, online latency-critical (LC) services are co-located with best-effort (BE) analytical jobs to improve resource utilization. The online services, such as requests from product search engine, online shopping and advertising services, are customer-facing and associated with strict Service Level Objectives (SLO). The instance of one online service is deployed in a stateful or stateless pod. Best-effort batch jobs, such as MapReduce and other data-intensive or computation-intensive analytical tasks, are submitted by internal users from different business units. These BE jobs are often treated as the "second-class citizens" and if necessary, they can be deferred or restarted to release resources for LC services.

2.2 Bursty Workloads

Bursty workloads caused by promotion campaigns are common for E-commerce giants and bring particular challenges for resource allocation in a data center. Besides the well-known Black Friday Sales and Double 11 Global Shopping Festival, there are many other sales promotion campaigns such as Women's Day Festival, 618 promotion[1] and Great Singapore Sale[2]. During the periods of these campaigns, customers flood in with an increasing volume of LC service requests. Besides the E-commerce promotion campaigns, a bursty workload pattern also exists in daily services, where the workload would suddenly increase by around 2-10 times. For example, requests for restaurant orders and food delivery services increase dramatically during the lunch and dinner periods. Other scenarios include sales promotions from online livestream broadcasts, where the bursting time of LC services is unknown in advance.

To guarantee the SLO of these online services, our data center currently adopts a relatively conservative strategy to suppress the

[1]https://www.chinainternetwatch.com/33642/618-shopping-festival-2022/
[2]https://shopee.sg/m/gss

BE jobs by fixing a low resource quota for them. Such a strategy results in a particularly poor utilization rate and we are motivated to develop more agile and effective resource allocation strategies to tackle bursty workloads and improve resource utilization during the promotion periods.

2.3 Dynamic Resource Allocation

There are multiple types of resources, such as CPU, memory, and network, that require appropriate management in a data center. In this paper, we consider CPU as the most important resource and follow [16] to focus on CPU resource allocation. The reason is that most data centers suffer from low CPU utilization and a slight improvement can yield considerable profit. For example, as pointed out by [14], it is observed that the BE jobs can only conservatively harvest limited amount of resources and the average resource utilization of CPU in Alibaba cloud is only 38.2%. In contrast, the memory utilization rate may be less painful for a data center. As reported in [14], the average utilization rate for memory resources has reached 88.2% in Alibaba cloud and is not the main performance bottleneck. Therefore, our research scope in this paper is focused on CPU resource allocation strategies in a private cloud with co-located and bursty workloads.

To dynamically adjust the CPU resources allocated to BE jobs, we can launch a container to run the assigned BE jobs and leverage the Linux command *cgroups* to manage the CPU quota of the container, i.e., the maximum CPU cycles that can be allocated to a container under the Completely Fair Scheduler (CFS) of Linux. Our proposed PREACT system runs as a daemon to perform QPS prediction and resource allocation tasks.

2.4 Problem Definition

Here we study the problem of CPU resource allocation in the scenario of bursty workloads. Our objective is to guarantee the SLO of LC services, and in the meanwhile, maximize the CPU utilization. The SLO is defined using the indicator of response time. For each LC service, we have a threshold η to measure the maximum amount of response time that is tolerable for the service users. If the response time exceeds η, we say the SLO constraint is violated. Note that when a bursting workload floods in, the SLO violation may still be inevitable even when all the CPU resources have been allocated to LC services. In the experimental study, we use both SLO violation rate and CPU utilization rate as two main performance metrics.

3 RELATED WORK

Resource management for data centers has been intensively studied. We focus on the review of dynamic resource managers for co-located workloads and roughly categorize them into feedback-based controllers and predictor-based controllers. The difference is that feedback-based controllers react promptly to the current observations, whereas predictor-based controllers are associated with a prediction module and make decisions based on future trends.

3.1 Feedback-based Controllers

Bistro [13] from Facebook runs data-intensive batch jobs on live production systems with online tenant workloads. It monitors the real-time resource usage at runtime and schedules appropriate tasks whenever extra resources become available. However, it requires manual intervention to adjust the resource quota for the batch jobs. Heracles [22] shares the same objective with our work to maximize resource utilization without violating the SLO constraint. It continuously monitors the latency slack, which refers to the difference between the SLO latency target of LC workloads and the actual latency observed. As long as the slack is found to be within a threshold, batch jobs will be killed to release resources. PARTIES [3] adopts a similar quota management strategy to Heracles by monitoring the latency slack and extends Heracles to support more LC services and resource types. PerfIso [16] from Microsoft preserves an additional buffer of idle cores to absorb any LC services that wake up. Its scheduler keeps track of the number of idle cores (excluding those in the buffer) and allocates them to BE jobs.

These feedback-based controllers rely on monitoring the specified indicators and adjusting the resource allocation according to the latest observations. When a large volume of service requests flood in, their response could be lagged when facing bursty workloads of LC services. The consequence is that they may fail to react promptly, and have a higher chance of incurring SLO violation.

More recently, emerged feedback-based controllers that can react to abrupt workload changes with low latency. Shenango [24] and Caladan [12] are such representative systems. Shenango considers queuing delays as the signals to decide if an LC service can benefit from more cores. If congestion is detected, its CPU core reallocation can be completed in $5.9\mu s$. Caladan is an extended work of Shenango. It uses multiple control signals to manage more types of interference as well as changes in load. However, for these fast reactive approaches, highly frequent resource adjustment may play a negative effect on the system performance and stability (e.g., frequent cache missing, overhead caused by lock acquisition and release), especially when the LC services require operations on the database. In our experiments, we will compare our PREACT with the fast reactive approaches when handling bursty workloads.

3.2 Predictor-based Controllers

Scavenger [17] is a predictive resource manager for batch workloads. Given a moving window of observations, it uses the statistical information (including the mean μ and standard deviation σ) of the LC services' resource usage, denoted by R_{LC}, to dynamically adjust the quota for the batch jobs. If R_{LC} exceeds $(\mu + c\sigma)$, Scavenger immediately pauses or kills a subset of batch jobs to release resources for the LC services. Autopilot [26] from Google can also be applied as a predictive resource manager for batch jobs. Similar to Scavenger, it is associated with a window-based prediction module to estimate the resource usage of LC services, and the remaining can be allocated to the batch jobs.

These statistical-based predictors are simple, lightweight, but not sufficiently accurate. On the one hand, they are agile to capture the dynamic workload pattern to a certain extent. On the other hand, it is difficult for them to react properly to bursty workloads. As will be evaluated in the experiments, these models generate considerable prediction errors when facing abrupt and bursty workloads. In this paper, we will propose a more accurate predictor to estimate the amount of incoming LC services in the next time window. In addition, we construct an SLO profile to capture the relationship

between the response time and incoming workload under different resource constraints. Given the predicted workload, we can find the maximum resources allocated to BE jobs without SLO violation.

3.3 Interference Minimization for Co-Located Workloads

There also exists a research branch whose objective is to minimize the interference between the LC services and BE batch jobs when they are competing for shared resources. For example, a considerable number of methods adopt isolation techniques at the hardware and OS levels to eliminate performance interference between different tasks [3, 9, 16–19, 21]. Though they can ensure the SLO of the LC services, these methods are conservative with low resource utilization rate. To further improve resource utilization, fine-grained workload deployment strategies [6–8, 10, 23, 33] are proposed to carefully isolate interference between the LC services and BE jobs, utilizing the hardware features and software isolation mechanisms.

The research attention of these works is the deployment strategy for the co-located workloads. It is complementary to our problem — when a set of LC services and BE jobs have been deployed to the same server, our focus is the dynamic resource allocation w.r.t. varying patterns of the incoming workload.

4 RESOURCE ALLOCATION MANAGER

In this section, we present our predictive resource allocation manager, namely PREACT. It consists of an agile and accurate predictor to estimate the future workload in the next time window and an SLO profile to capture the relationship between the response time and incoming workload under different resource constraints. Next, we will introduce the implementation details of these two modules. The frequently used notations are summarized in Table 1.

Table 1: Notations.

$QPS(t)$	The maximum number of incoming queries per second in the t-th time window
$g(t)$	Normal workload in the t-th time window
$e(t)$	Bursty workload in the t-th time window
(w, b)	Parameters for linear regression to estimate $g(t)$
$\mathbb{I}_b$	Binary indicator for bursty event
CPU_{lc}	The number of CPU cores allocated to LC services
RT	Response time
μ	The number of requests that can processed in a time unit
P_N	The probability of a request being discarded

4.1 QPS Predictor for LC Services

There have been various models proposed for time series forecasting, including traditional statistical approaches [11, 28] and recent deep learning models [27, 34]. These models are good at capturing periodic or stable patterns, but fail to work well for highly dynamic patterns. In this paper, we adopt a decomposable time series model (DTSM) [15] for bursty workload prediction. Its overall idea is to decompose the target value into multiple components, develop individual prediction models for each component, and aggregate

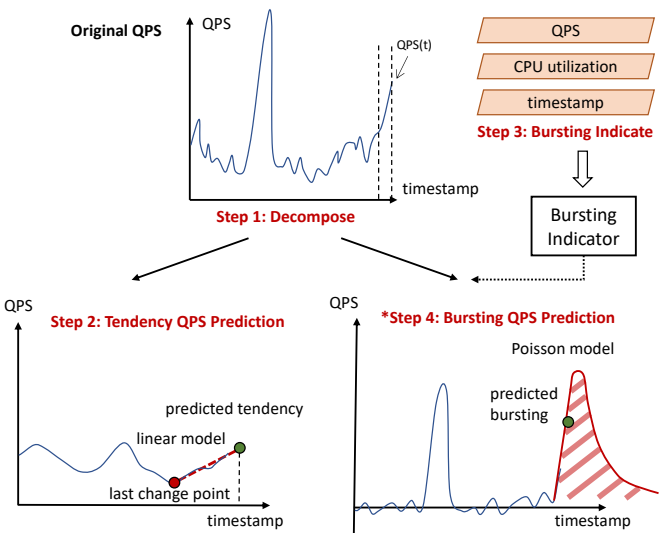


Figure 2: QPS prediction process. In step 1, the QPS signal is decomposed into a tendency signal $g(t)$ and a busty signal $e(t)$. In step 2, $g(t)$ is predicted using Equation 2. In step 3, a binary SVM classifier is to generate bursting event indicator I_b. In step 4, if I_b is positive, $e(t)$ is predicted using Equation 4.

their outputs as the final result. Compared with deep learning approaches, DTSM is interpretable and its decomposed components can be customized to capture irregular patterns. In addition, its model fitting is fast and suitable for the dynamic change in the bursty workload pattern.

As shown in Figure 2, we split the temporal dimension into equal-size time windows, whose length determines the frequency of our resource allocation strategy triggered to adjust the number of CPUs allocated to the BE jobs. Let $QPS(t)$ denote the maximum number of incoming queries per second in the t-th time window. We decompose $QPS(t)$ into two components:

$$QPS(t) = g(t) + e(t) \tag{1}$$

Here, $g(t)$ represents the workload trend in normal days and $e(t)$ is the event signal for bursty workloads.

4.1.1 Forecasting g(t). Following Facebook's Prophet [28], we adopt its piecewise linear regression model to simulate the varying pattern of $g(t)$. It first automatically identifies the set of change points from the observed data in the past time windows. These change points indicate the reverse of increase or decline pattern and serve as the endpoints of the piecewise line segments. Let t be the current time window and s_j be the closest change point with $s_j < t$. The observation records within the period $[s_j, t]$ are fitted by linear regression:

$$g(t) = \omega t + b \tag{2}$$

The optimal parameters for ω and b are calculated via the least squares method. Afterward, we can apply the linear regression model to predict $g(t+1) = \omega(t+1) + b$.

4.1.2 Forecasting e(t). With the prior knowledge of sales promotion campaigns, it is an easy task to determine the starting time of

the bursty workload. Nonetheless, to build a more general model to handle unforeseen bursts, we do not take advantage of the prior knowledge. Instead, we develop a lightweight binary classifier with SVM to indicate whether a bursty event occurs. We use $QPS(t-1)$, CPU utilization rate and timestamp to form an input vector for binary classification. The reason to select CPU utilization is that a bursty event is likely to incur high CPU utilization rate and it can be used as a complementary clue to QPS. The timestamp is represented by a 6-dimensional vector to capture the information of year, month, day, hour, minute and second.

To train the binary SVM classifier, we collect the time series information of CPU utilization in the past promotion events as the positive samples. The negative samples are generated from daily workload data without bursty events. To estimate $e(t)$, we first run the binary classifier for bursty event detection. Once the indicator, denoted by $\mathbb{I}_b$, becomes positive, we need to predict the value of bursty workload $b(t)$ and use the derived $e(t)$ in the following equation to update $QPS(t)$.

$$e(t) = \mathbb{I}_b \cdot b(t) \tag{3}$$

Since bursty workload results from a large number of random requests from users within a short period of time, we model $b(t)$ using Poisson distribution, which was applied to generate bursty workload for cloud computing [32]. Let B refer to the total number of additional requests that will arrive in the period with bursty workload, i.e., the time interval with positive $\mathbb{I}_b$. We apply Poisson distribution and use the following equation to model $b(t)$, which represents the estimated number of requests arriving in the t-th time window.

$$b(t) = B \cdot \frac{\lambda^t e^{-\lambda}}{t!} \tag{4}$$

Here, B is unknown in advance and we treat B and λ as two parameters in the model to be learned from the observation data. More specifically, we apply the maximum likelihood estimation to train these two parameters.

4.1.3 Model Training. The SVM classifier is a standalone module that is trained once from historical data to generate binary indicator I_b. The DTSM model, excluding the SVM classifier, consists of parameters ω and b in Equation 2, B and λ in Equation 4. The model is re-trained in every time window, whose length is set to one minute in our implementation. The effect of different window sizes will be evaluated in the experiments. The observations of $QPS(t)$ in the past 10 minutes are used to generate training data. Given the sequence of observed $QPS(t)$, we use the following three steps to decompose each $QPS(t)$ into two signals and regress $g(t)$ and $e(t)$ separately.

(1) Apply median filter [2] on the sequence of $QPS(t)$ to remove outliers in the sequence.
(2) Use moving average smoothing [25] on the sequence again to further reduce noise and obtain smoothed tendency QPS for $g(t)$.
(3) Set the signal $e(t)$ as the residual between $QPS(t)$ and $g(t)$.

From the derived sequence of $g(t)$ in the past time windows, we first detect the last change point and then apply linear regression to learn parameters in Equation 2. For the sequence of $e(t)$, we apply the maximum likelihood estimation to learn parameters in Equation 4 if the indicator $\mathbb{I}_b$ is positive. In addition, we note that with the same amount of prediction error, overestimation is preferred over underestimation. This is because overestimation of QPS in the next time window only leads to lower resource utilization, but underestimation of QPS may cause SLO violation. Therefore, we modify the loss function of $g(t)$ and $e(t)$ and add a penalty factor to punish underestimation Equation 5 and 6.

$$\min_{\omega,b} \sum_{i=1}^{n} (y_i - g(\hat{t}))^2 + \beta \max\{y_i - g(\hat{t}), 0\} \tag{5}$$

$$\min_{B,\lambda} \sum_{i=1}^{n} (y_i - b(\hat{t}))^2 + \beta \max\{y_i - b(\hat{t}), 0\} \tag{6}$$

4.2 SLO Profiling Model

Recall that in our problem definition, the SLO of an LC service is determined by the indicator of response time (RT). As long as its RT is smaller than a pre-defined threshold η, we say that the SLO is not violated. There are two key factors that can affect the response time of an LC service, including the QPS of LC services and the number of CPU cores allocated to process these online requests, denoted by CPU_{lc}. It is straightforward that when QPS increases dramatically when bursty workload arrives, we need to increase CPU_{lc} promptly in order to satisfy the SLO requirement. Once we have predicted $QPS(t)$ in the next time window, our goal is essentially to infer the minimum setting of CPU_{lc} such that when CPU_{lc} cores are allocated to the LC service, its response time does not exceed η. This is equivalent to maximizing the number of CPU cores allocated to the BE jobs.

To find the minimal CPU_{lc} without SLO violation, we propose an SLO profiling strategy with a parametric model based on queuing theory and optimize it with neural networks. The objective is to build a mapping function $M(QPS, CPU_{lc}) \rightarrow RT$ as the RT profile such that given a predicted value of $\hat{QPS}$, we can conveniently lookup from the function to find the minimum CPU_{lc} with $M(\hat{QPS}, CPU_{lc}) < \eta$.

To learn the mapping function $M(QPS, CPU_{lc})$, we model the query processing of LC services as a multi-server queuing model [5], from which we can approximate the response time using the following equation[1]:

$$RT \approx \frac{1}{\mu} \frac{1}{(1 - ((1 - P_N)\rho)^m)} \tag{7}$$

Here, μ is the number of requests that can be processed in a time unit. P_N captures the probability of a request being discarded when the queue is full. When $P_N > 0$, it implies that the bursty workload cannot be handled in real-time and a small portion of requests will be discarded without further processing. ρ is a parameter that depends on μ and QPS and it is defined as $\rho = \frac{QPS}{\mu}$.

Since it is difficult to explicitly determine the parameters in Equation 7 (e.g., μ is implicitly dependent on both QPS and CPU_{lc}), we resort to adopting neural networks for RT prediction. More specifically, we build two MLP (multi-layer perceptron[3]) models to learn parameters μ and P_N. These two parameters are dependent on the number of requests QPS and the available CPU cores CPU_{lc}.

[3] https://en.wikipedia.org/wiki/Multilayer_perceptron

Thus, we concatenate them as the input of the MLP models.

$$\mu = MLP_1(QPS, CPU_{lc}) \quad (8)$$

$$P_N = MLP_2(QPS, CPU_{lc}) \quad (9)$$

In our implementation, MLP_1 and MLP_2 have the same structure but they are trained separately. There are two hidden layers and each layer contains 8 nodes. To train the MLP models, we conduct offline experiments to obtain training samples. Each sample is a triplet $\langle QPS, CPU_{lc}, RT\rangle$ in which RT is the ground-truth label. The neurons in the two MLPs are trained with the standard back propagation algorithm. Once these two MLP models are trained, we can use them to obtain the predicted values of μ and P_N and apply them in Equation 7 to estimate RT.

4.3 Resource Allocation Algorithm

To piece together the aforementioned modules, we summarize the procedure of our resource allocation algorithm in Algorithm 1. The allocation manager runs as a daemon and is activated in each sliding time window. We collect the recent QPS records in the past 10 minutes and decompose the observations of $QPS(t)$ to extract trending and event signals for model training. The linear model $g(t)$ is re-trained from the observations. If the binary classifier for bursty workload detection returns a positive indicator $\mathbb{I}_b$, we learn $e(t)$ from the Poisson model in Equation 4. With the updated model, we predict $QPS(t+1)$ in the next time window. From the mapping function constructed offline, we scan the entries whose key contains $QPS(t+1)$ and find the minimum CPU_{lc} with the estimated RT smaller than η. Finally, we update the maximum number of CPU cores allocated to BE jobs as $CPU_{total} - CPU_{lc}$, where CPU_{total} is the number of CPU cores in the server.

Algorithm 1 Predictive Resource Allocation Algorithm

```
for each time window t = 1, 2, 3, ... do
    Run the SVM classifier and obtain the binary indicator 𝕀_b for the
    event of bursty workload
    Obtain the historical QPS records in the past 10 minutes
    Re-train the linear model g(t) in Equation 2
    if 𝕀_b = 1 then
        Learn Poisson model b(t) from Equation 4
        e(t) ← b(t)
    end if
    Predict QPS(t + 1) in next time window
    CPU_lc ← 0
    for C ← 1; C < CPU_total; C++ do
        if M(QPS(t + 1), C) < η then
            CPU_lc ← C
            break
        end if
    end for
    Allocate CPU_total − CPU_lc cores to BE jobs
end for
```

5 EXPERIMENT EVALUATION

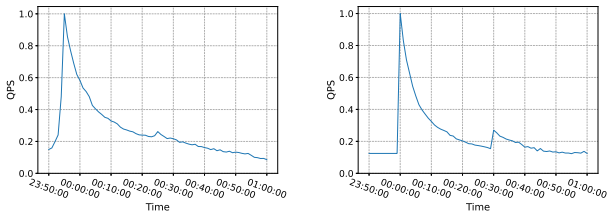
In this section, we first present the experimental setup with co-located workloads, comparison methods and performance metrics. Then, we report the overall performance of PREACT and validate its superiority over its competitors. Finally, we conduct a break-down analysis on PREACT to evaluate the performance of its two core modules — QPS predictor and SLO profiling.

5.1 Experimental Setup

Co-located Workload Generation. For LC services, we select two representative E-commerce queries to generate online requests that need to be processed in real-time, including

- **Cart** service, which refers to adding a candidate item into the shopping cart. We set its maximum tolerable response time to be 150ms and use it as the default SLO.
- **Buy** service, in which the order is confirmed by the customer and ready for payment. Its SLO is set to 175ms. If its response time is larger than 175ms, we consider an SLO violation incurred.

We use the real trace logs of a major E-commerce promotion campaign to generate bursty workloads for these two LC services. As shown in Figure 3, we select the period from 23:50pm to 01:00am[4] and plot their normalized QPS values during this period. Note that the QPS of the Cart service is significantly higher than the Buy service before normalization. For the Buy service, there is an abrupt and sharp increase pattern starting from 00:00am, followed by a dramatic decline in the next 10 minutes. The Cart service also demonstrates a similar pattern, except that its increasing pattern is not as abrupt as the Buy service. This is because the customers tend to put their candidate items in the shopping cart before 00:00am and finish the order after 00:00am so as to enjoy the promotion discount.



(a) Cart (b) Buy

Figure 3: QPS pattern of two LC services.

For simplicity, we use **Spark-Bench** [20] to generate BE jobs. It consists of a comprehensive suite of workloads, including analytical tasks from machine learning, graph computing, SQL query, and streaming applications. In other words, Spark-Bench provides a good simulation of our real analytical tasks. It also saves the troublesome procedures of applying for access grants for real BE jobs and coordinating with other business units during the experiments. These tasks are computation-intensive and the details of Spark-Bench are presented in Table 2. The workload is randomly selected when there are available CPU cores for BE jobs.

The server in our private cloud is equipped with 104 Cascade Lake-based logic cores (Intel(R) Xeon(R) Platinum 8269CY CPU @ 2.50GHz) and 370GB memory size. Since our experiments only involve two LC services, we use one server to handle the Buy service and the other for the Cart service.

[4]For anonymity, the exact date is not reported to avoid revealing the affiliation information.

Table 2: Spark-Bench Workloads

Category	Workload
Machine Learning	Logistic Regression/SVM/FM
Graph Computation	PageRank/SVD++/TriangleCount
SQL queries	Hive/RDDRelation
Streaming Application	Twitter/PageView

Table 3: The overall performance w.r.t. two LC services.

Methods	Cart		Buy	
	CPU	Violation	CPU	Violation
Perfiso	71.43%	14.29%	73.34%	14.29%
Autopilot	72.98%	14.29%	77.64%	10.00%
Scavenger	61.11%	12.86%	60.89%	7.14%
PREACT	**80.21%**	**8.57%**	**84.71%**	**5.71%**

Comparison Methods. We compare PREACT with the feedback-based controller Perfiso [16] from Microsoft and the predictor-based controllers, including Scavenger [17] and Google's Autopilot [26]. Their resource allocation strategies for co-located workloads have been presented in Section 3.

Performance Metrics. We set the length of sliding time window to be 1 minute, because the QPS statistics of LC services are collected every 1 minute in our production cloud. By default, the resource allocation manager is activated every one minute. Our ultimate goal is to maximize the resource utilization rate without SLO violation. Therefore, we need to take into account the CPU utilization rate and the frequency of SLO violation as two key performance metrics. Since the online requests of LC services in our experiment last for 70 minutes (as shown in Figure 4), we collect the CPU utilization rate via Linux system command at the frequency of every minute and report the average value. To quantitatively measure the SLO violation, we need to determine whether the response time exceeds the threshold η. Since there are a large number of online queries processed in a sliding time window, we follow previous works [16, 17] to use the 95^{th} percentile as $RT(t)$. The violation rate is then defined as:

$$\text{violation rate} = \frac{\sum_t \mathbb{I}(RT(t) > \eta)}{T}$$

Here, $\mathbb{I}(RT(t) > \eta) = 1$ if $RT(t) > \eta$. Otherwise, $\mathbb{I}(RT(t) > \eta) = 0$. T equals to 70 in our experimental setup.

Processing Overhead. PREACT is lightweight and resource friendly. The time series prediction model only involves several parameters. As to memory consumption, we maintain a small window of size 10 to store the information of QPS, CPU utilization, etc, and the size of the trained SLO profiling model is less than 2KB. The window-based training time and prediction time are 0.339 seconds and 0.041 seconds, respectively.

5.2 Overall Performance

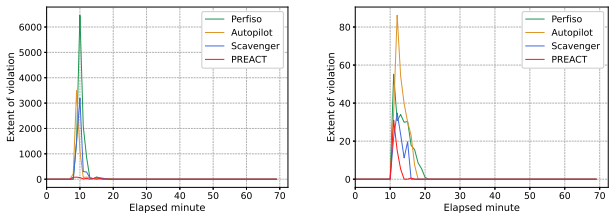
In the first experiment, we compare PREACT with Perfiso, Autopilot, and Scavenger and report their CPU utilization rate and SLO violation rate in Table 3. We can see that in both LC services, PREACT achieves the highest CPU utilization rate and the lowest violation rate at the same time. Such superiority validates the effectiveness of our proposed QPS predictor and SLO profiling strategy to tackle bursty workloads with highly dynamic patterns. Between the two LC services, Cart results in a larger violation rate because its QPS is significantly higher than Buy service and only a portion of the add-to-cart actions will be converted to final payment.

Among the comparison approaches, the two predictive methods, including Autopilot and Scavenger, achieve a lower SLO violation rate than Perfiso, which is a feedback-based controller. This is because the predictive approach can forecast the increasing pattern in the next time window and pre-allocate more resources for LC services. In contrast, even though Perfiso is associated with a fixed-size buffer reserved for LC services, it is lagged in decision making and cannot promptly handle the bursty workload. Autopilot results in a higher violation rate than Scavenger because it applies moving average to predict the CPU utilization rate of LC service in the next time window. Such prediction is not accurate in front of a sudden and sharp increase pattern. With a higher violation rate, it in fact allocates more CPU cores to BE jobs and obtains higher resources than Scavenger.

5.3 Detailed Visualization and Analysis

In this part, we present a more detailed perspective on how resource managers react to the varying patterns of LC services. The objective is to reveal more details and justify the reasons for the superiority of PREACT.

In Figure 4, we plot the extent of violation at each time window, which is defined as $\frac{RT(t)-\eta}{\eta}$ when $RT(t) > \eta$. If there is no violation, the value is set to 0. This indicator provides more detailed information than the overall violation rate in Table 3. From the plotted results, we can see that in the initial 10 minutes, the violation rates of all the methods are 0 because the QPS of LC service remains at a low level and can be processed instantly. The violation occurs when the bursty workload arrives. Since the QPS of Cart is significantly higher than the Buy service, its violation extent is much worse.

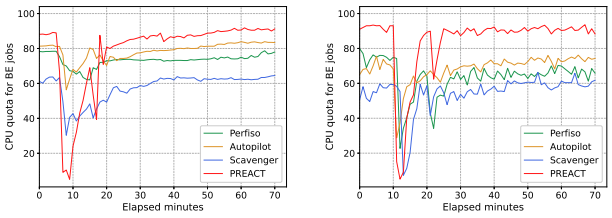


(a) Cart service (b) Buy service

Figure 4: Extent of violation in each time window.

Among other comparison methods, Perfiso maintains a buffer of idle CPU cores. Thus, its assigned number of BE jobs is not as high as PREACT even in the time windows with less dramatic workload variation. Autopilot outperforms Scavenger because its predictor is more sensitive to workload change. We also note that our PREACT allocates more resources than Autopilot and Scavenger in the non-bursty period. Our explanation is that in this period, the workload

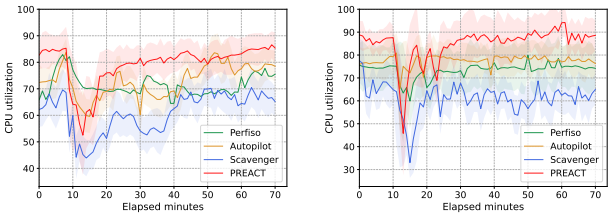
still demonstrates a decline pattern. Our predictor is more accurate and can pre-allocate more resources Figure 5.



(a) Quota allocation of Cart service (b) Quota allocation of Buy service

Figure 5: Quota allocation in each time window.

In Figure 6, we plot the average CPU utilization rate in each time window. We can see that in the bursty period, the utilization rate of PREACT is not the best around the peak workload. It chooses to sacrifice the CPU utilization rate to prioritize the requirement of SLO. Nonetheless, in the other time windows without SLO violation, PREACT can still outperform the other approaches in terms of CPU utilization rate, which helps PREACT to achieve the highest CPU utilization rate and the lowest SLO violation rate in Table 3.



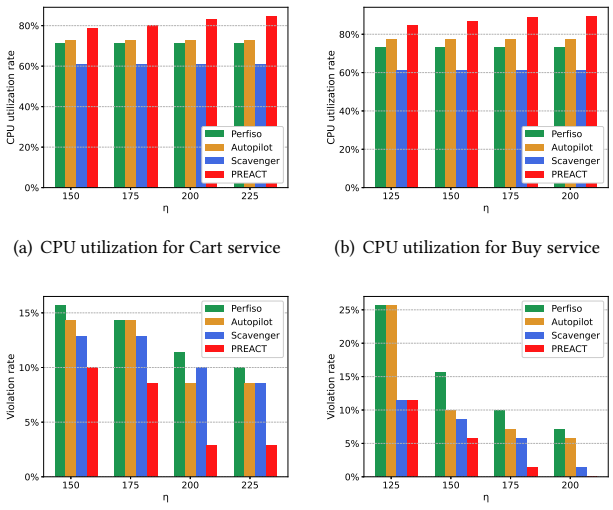
(a) CPU utilization of Cart service (b) CPU utilization of Buy service

Figure 6: CPU utilization in each time window.

5.4 Adaptivity to RT Threshold

A desirable feature of PREACT is that they can be adaptive to different settings of RT thresholds. In other words, it allows the service providers to dynamically adjust the maximum amount of tolerable response time. In contrast, Perfiso adjusts the resource allocation simply based on the buffer condition. As long as LC service starts to occupy the buffered CPU, it considers that the current resource allocation may not be safe and suppresses the BE jobs to release resources. Autopilot and Scavenger directly predict the CPU utilization rate of the LC service from the observations in recent time windows. The underlying intuition is CPU utilization is positively correlated with the QPS of LC service. From the regression on the CPU utilization rate, they can directly determine the amount of resources required by the LC service. Thus, the update of RT threshold will not affect their decision making for resource allocation.

We can see from Figure 7 that when RT threshold η increases from 125ms to 200ms, the CPU utilization of PREACT increases as well because it can adopt a more aggressive strategy to assign a higher number of CPU resources to the BE jobs without violating the SLO constraint. The CPU utilization of three competitors remain the same because their allocation strategies do not take into account η as an input. As to the violation rate, we can observe a clear decreasing pattern for all the methods. This is because when η increases, it becomes less likely to violate the SLO constraint $RT(t) < \eta$.



(a) CPU utilization for Cart service (b) CPU utilization for Buy service

(c) Violation rate for Cart service (d) Violation rate for Buy service

Figure 7: Performance with varying RT.

5.5 Break-down Analysis

Since PREACT incorporates QPS prediction and SLO profiling as two key contributions, we conduct a break-down analysis to evaluate their performance respectively.

5.5.1 Accuracy of Time Series Prediction Models. To demonstrate the superiority of our QPS predictor, we compare it with multiple time series forecasting methods, including

- **EMA** (Exponential Moving Average)[5], a type of moving average (MA) that applies exponentially decreasing weighting factors. The method is used in AutoPilot.
- **ARIMA** (Autoregressive Integrated Moving Average) [11], a kind of autoregressive model widely used in time series forecasting.
- **DSLTM** [27], a deep neural network with multiple LSTM layers.
- **Informer** [34], which applies Transformer [30] architecture and is considered as the state-of-the-art deep learning model for long sequence forecasting problems.
- **Prophet** [28] and **Neural Prophet** [29] from Facebook, which belongs to the category of decomposable time series model (DTSM). Prophet decomposes time series into tendency, season, and event. Neural Prophet further extends Prophet with autoregression.

[5]https://en.wikipedia.org/wiki/Moving_average

Among these approaches, DSLTM and Informer are based on deep learning and require additional training datasets for supervised training. The remaining approaches and our PREACT only require the observations in recent time windows to update model parameters. To construct training datasets for DSLTM and Informer, we use the workload data from the promotion campaigns in previous years and construct around 7,000 labeled records. Then, we use cross-validation to train these two models, whose implementations are available on github[6]. We compare the performance of these time series predictors using MAPE and MSE.

We run the time series prediction models using the trace log of Buy service as test data and report prediction errors in Table 4. The traditional statistical approaches such as EMA and ARIMA outperform the deep learning models, including DLSTM and Informer. This finding is not surprising because deep learning models are good at learning regular patterns from a large amount of training data, but they are not accurate enough to predict the irregular bursty workload with high dynamism. For the prediction of bursty workload, there lack of sufficient training data from past years and the pattern could vary significantly. Both issues bring challenges to the convergence of the training process. In contrast, EMA and ARIMA only rely on recent observations and are sensitive to the dynamic update of workload trends.

Among the decomposable time series models, Neural Prophet is slightly better than Prophet because it extends the basic Prophet model with a feed-forward neural network for data regression. The feed-forward neural network is considered to be more expressive than linear model regression. Nevertheless, these two models are both inferior to our PREACT. The reason is that our QPS predictor is more powerful in capturing the bursty event signal $e(t)$. We have a binary classifier based on SVM for bursty event detection and a Poisson model to learn the pattern of bursty workload.

Table 4: QPS prediction results for Buy service.

Type	Algorithm	MAPE	MSE
Traditional Models	EMA	26.21%	2983.5
	ARIMA	19.97%	2482.1
Deep Learning Models	DLSTM	47.18%	9931.7
	Informer	40.01%	7177.3
Decomposable Models	Prophet	22.93%	2671.0
	Neural Prophet	18.95%	1923.8
	PREACT	**13.58%**	**1183.9**

5.5.2 Performance of SLO Profiling. Next, we evaluate the mapping function $M(QPS, CPU_{lc}) \rightarrow RT$. We apply queuing theory to build a parametric model and optimize it with neural networks. To collect training samples, we decompose the input space into 11×15 grid cells, with a step size of 2,000 for QPS and 10 for CPU cores. For each combination of (QPS, CPU_{lc}), we run the experiments three times and obtain three RT values as the ground-truth labels. We expect the model to automatically learn a good regression to minimize the total errors in all these labeled data. In Table 5, we report the regression error in terms of MAPE and the time cost for training and inference. The training process is very fast and it takes less than 42 seconds to train each MLP model to approximate the mapping function $M(QPS, CPU_{lc}) \rightarrow RT$. The inference time for one-time prediction is around 0.035 ms. In the worst case, it takes at most 3.64ms to identify the maximum resources allocated to BE jobs (lines 11-15 in Algorithm 1).

[6]https://github.com/DeepWolf90/DLSTM2
https://github.com/zhouhaoyi/Informer2020

Table 5: Performance of SLO Profiling.

Services	MAPE	Training Time	Inference Time
Cart	4.69%	41.9 s	0.035 ms
Buy	5.41%	40.5 s	0.034 ms

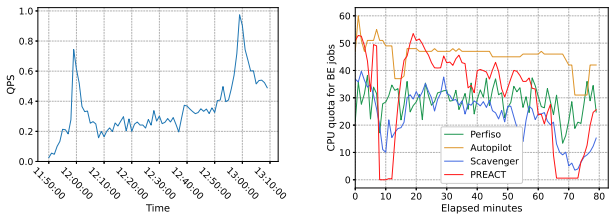
5.5.3 Ablation Study. We also conduct an ablation study to reveal the contributions of QPS predictor and SLO profiling. The first variant of PREACT is constructed by replacing its QPS predictor with ARIMA algorithm. The second variant replaces the SLO profiling with linear regression to capture the relationship between QPS and CPU quota. We can see from Table 6 that the accuracy of QPS predictor plays an important role for violation rate. When it is replaced with ARIMA, the violation rate increases considerably. On the other hand, with an accurate QPS predictor, the component of SLO profiling can help increase the overall CPU utilization.

Table 6: Ablation study using Cart service.

	PREACT	First variant	Second variant
CPU rate	80.21%	80.67%	70.24%
Violation rate	8.75%	12.86%	8.67%

5.6 Experiment on Other Workload Pattern

In the final experiment, we conduct experiments to evaluate the performance of PREACT under a different bursty workload pattern. We choose an LC service that is responsible for customer voucher management. In the days of the promotion campaign, a certain amount of vouchers will be offered to customers hourly on a first-come-first-served basis. As shown in Figure 8(a), we plot the workload of voucher service from 11:50am to 13:10pm and there are two peaks at the beginning of each hour. Its pattern of workload variation is not as dramatic as the Cart and Buy services in Figure 3. In addition, the workload shows an increasing trend in this period.



(a) Workload pattern (b) Quota allocation for BE jobs

Figure 8: Workload pattern and quota allocation for Voucher services.

Table 7: The overall performance w.r.t. vouch service.

Methods	CPU Utilization	SLO Violation
Perfiso	85.42%	7.75%
Autopilot	**91.47**%	18.75%
Scavenger	83.68%	2.50%
PREACT	90.83%	**0.00**%

In Table 7, we report the overall performance in terms of CPU utilization rate and SLO violation rate, with RT threshold set to 225ms. In the two peaks with the bursty workload, since the increasing pattern is less dramatic than Cart and Buy services, PREACT pre-allocates sufficient CPU resources to voucher service and incurs no SLO violation. Its CPU utilization rate is slightly lower than that of Autopilot. The reason is that Autopilot sacrifices SLO violation for high CPU utilization. As shown in Figure 8(b), in the time window of [40, 80] minutes, Scavenger and PREACT allocate fewer CPU cores to BE jobs as the workload of voucher service continues to grow. However, Autopilot still maintains high level of quota allocation for BE jobs in this period, which results in the most severe SLO violation.

6 CONCLUSION AND FUTURE WORKS

In this paper, we proposed PREACT as a predictive and agile resource allocation manager to support bursty workloads in a co-located data center. We implemented an accurate and lightweight predictor based on a decomposable time series model for future QPS estimation. We also devised an SLO profiling model based on queuing theory to determine the optimal resource allocation. Experiments are conducted using real LC services in our data center, with the workload pattern collected from a major E-commerce promotion campaign. The results show that PREACT can achieve a low SLO violation rate and high resource utilization simultaneously.

ACKNOWLEDGMENTS

This work was supported by the Fundamental Research Funds for the Central Universities 226-2024-00145, Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security, and Alibaba Group through Alibaba Innovative Research Program.

REFERENCES

[1] Ian Angus. 2001. An introduction to Erlang B and Erlang C. *Telemanagement* 187 (2001), 6–8.
[2] GONZALOR Arce and MICHAELP McLoughlin. 1987. Theoretical analysis of the max/median filter. *IEEE transactions on acoustics, speech, and signal processing* 35, 1 (1987), 60–69.
[3] Shuang Chen, Christina Delimitrou, and José F. Martínez. 2019. PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services. In *ASPLOS*. ACM, 107–120.
[4] Yue Cheng, Zheng Chai, and Ali Anwar. 2018. Characterizing Co-located Datacenter Workloads: An Alibaba Case Study. In *APSys*. ACM, 12:1–12:3.
[5] Robert B Cooper. 1981. Queueing theory. In *Proceedings of the ACM'81 conference*. 119–122.
[6] Christina Delimitrou and Christos Kozyrakis. 2013. QoS-Aware scheduling in heterogeneous datacenters with paragon. *ACM Trans. Comput. Syst.* 31, 4 (2013), 12:1–12:34.
[7] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: resource-efficient and QoS-aware cluster management. In *ASPLOS*. 127–144.
[8] Christina Delimitrou and Christos Kozyrakis. 2016. HCloud: Resource-Efficient Provisioning in Shared Cloud Systems. In *ASPLOS*. ACM, 473–488.
[9] Christina Delimitrou and Christos Kozyrakis. 2017. Bolt: I Know What You Did Last Summer... In The Cloud. In *ASPLOS*. ACM, 599–613.
[10] Christina Delimitrou, Daniel Sánchez, and Christos Kozyrakis. 2015. Tarcil: reconciling scheduling speed and quality in large shared clusters. In *SoCC*. ACM, 97–110.
[11] Jamal Fattah, Latifa Ezzine, Zineb Aman, Haj El Moussami, and Abdeslam Lachhab. 2018. Forecasting of demand using ARIMA model. *IJEBM* 10 (2018), 1847979018808673.
[12] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. 2020. Caladan: Mitigating Interference at Microsecond Timescales. In *OSDI*. USENIX Association, 281–297.
[13] Andrey Goder, Alexey Spiridonov, and Yin Wang. 2015. Bistro: Scheduling Data-Parallel Jobs Against Live Production Systems. In *ATC*. USENIX Association, 459–471.
[14] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. 2019. Who limits the resource efficiency of my datacenter: an analysis of Alibaba datacenter traces. In *IWQoS*. ACM, 39:1–39:10.
[15] Andrew Harvey and Simon Peters. 1990. Estimation procedures for Structural Time Series Models. *Journal of Forecasting* 9 (1990), 89–108.
[16] Calin Iorgulescu, Reza Azimi, Youngjin Kwon, Sameh Elnikety, Manoj Syamala, Vivek R. Narasayya, Herodotos Herodotou, Paulo Tomita, Alex Chen, Jack Zhang, and Junhua Wang. 2018. PerfIso: Performance Isolation for Commercial Latency-Sensitive Services. In *ATC*. USENIX Association, 519–532.
[17] Seyyed Ahmad Javadi, Amoghavarsha Suresh, Muhammad Wajahat, and Anshul Gandhi. 2019. Scavenger: A black-box batch workload resource manager for improving utilization in cloud environments. In *Proceedings of the ACM Symposium on Cloud Computing*. 272–285.
[18] Harshad Kasture, Davide B. Bartolini, Nathan Beckmann, and Daniel Sánchez. 2015. Rubik: fast analytical power management for latency-critical systems. In *MICRO*. ACM, 598–610.
[19] Harshad Kasture and Daniel Sánchez. 2014. Ubik: efficient cache sharing with strict qos for latency-critical workloads. In *ASPLOS*. ACM, 729–742.
[20] Min Li, Jian Tan, Yandong Wang, Li Zhang, and Valentina Salapura. 2015. SparkBench: a comprehensive benchmarking suite for in memory data analytic platform Spark. In *CF*. ACM, 53:1–53:8.
[21] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. 2014. Towards energy proportionality for large-scale latency-critical workloads. In *ISCA*. IEEE Computer Society, 301–312.
[22] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: improving resource efficiency at scale. In *ISCA*. ACM, 450–462.
[23] Jason Mars and Lingjia Tang. 2013. Whare-map: heterogeneity in "homogeneous" warehouse-scale computers. In *ISCA*. ACM, 619–630.
[24] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving High CPU Efficiency for Latency-sensitive Datacenter Workloads. In *NSDI*, Jay R. Lorch and Minlan Yu (Eds.). USENIX Association, 361–378.
[25] Aistis Raudys, Vaidotas Lenčiauskas, and Edmundas Malčius. 2013. Moving averages for financial data smoothing. In *International Conference on Information and Software Technologies*. Springer, 34–45.
[26] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. 2020. Autopilot: workload autoscaling at Google. In *EuroSys*. ACM, 16:1–16:16.
[27] Alaa Sagheer and Mostafa Kotb. 2019. Time series forecasting of petroleum production using deep LSTM recurrent networks. *Neurocomputing* 323 (2019), 203–213.
[28] Sean J Taylor and Benjamin Letham. 2018. Forecasting at scale. *The American Statistician* 72, 1 (2018), 37–45.
[29] Oskar Triebe, Hansika Hewamalage, Polina Pilyugina, Nikolay Laptev, Christoph Bergmeir, and Ram Rajagopal. 2021. NeuralProphet: Explainable Forecasting at Scale. *arXiv preprint arXiv:2111.15397* (2021).
[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[31] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *EuroSys*. ACM, 18:1–18:17.
[32] Jianwei Yin, Xingjian Lu, Xinkui Zhao, Hanwei Chen, and Xue Liu. 2015. BURSE: A Bursty and Self-Similar Workload Generator for Cloud Computing. *IEEE Trans. Parallel Distributed Syst.* 26, 3 (2015), 668–680.
[33] Laiping Zhao, Yanan Yang, Kaixuan Zhang, Xiaobo Zhou, Tie Qiu, Keqiu Li, and Yungang Bao. 2020. Rhythm: component-distinguishable workload deployment in datacenters. In *EuroSys*. ACM, 19:1–19:17.
[34] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of AAAI*.