Understanding and Optimizing Workloads for Unified Resource Management in Large Cloud Platforms

Chengzhi Lu*‡
Shenzhen Institute of Advanced Technology, CAS
Univ. of CAS, Univ. of Macau
Macau SAR, China
cz.lu@siat.ac.cn

Huanle Xu*‡
University of Macau
Macau SAR, China
huanlexu@um.edu.mo

Kejiang Ye‡
Shenzhen Institute of Advanced Technology, CAS
Shenzhen, China
kj.ye@siat.ac.cn

Guoyao Xu
Alibaba Group
Hangzhou, China
yao.xgy@alibaba-inc.com

Liping Zhang
Alibaba Group
Hangzhou, China
liping.z@alibaba-inc.com

Guodong Yang
Alibaba Group
Hangzhou, China
luren.ygd@taobao.com

Chengzhong Xu†‡
University of Macau
Macau SAR, China
czxu@um.edu.mo

Abstract

To fully utilize computing resources, cloud providers such as Google and Alibaba choose to co-locate online services with batch processing applications in their data centers. By implementing unified resource management policies, different types of complex computing jobs request resources in a consistent way, which can help data centers achieve global optimal scheduling and provide computing power with higher quality. To understand this new scheduling paradigm, in this paper, we first present an in-depth study of Alibaba's unified scheduling workloads. Our study focuses on the characterization of resource utilization, the application running performance, and scheduling scalability. We observe that although computing resources are significantly overcommitted under unified scheduling, the resource utilization in Alibaba data centers is still low. In addition, existing resource usage predictors tend to make severe overestimations. At the same time, tasks within the same application behave fairly consistently, and the running performance of tasks can be well-profiled with respect to resource contention on the corresponding physical host.

Based on these observations, in this paper, we design Optum, a unified data center scheduler for improving the overall resource utilization while ensuring good performance for each application. Optum formulates an optimization problem to schedule unified task requests, aiming to balance the trade-off between utilization and resource contention. Optum also implements efficient heuristics to solve the optimization problem in a scalable manner. Large-scale experiments demonstrate that Optum can save up to 15% of resources without performance degradation compared to state-of-the-art unified scheduling schemes.

*Co-first author. Both authors contributed equally to this paper.
†Corresponding author.
‡C. Lu, H. Xu, K. Ye and C. Xu are also with Guangdong-Hong Kong-Macao Joint Laboratory of Human-Machine Intelligence-Synergy Systems.

CCS Concepts: • **Computer systems organization** → **Cloud computing**.

Keywords: cloud computing, unified scheduling, resource over-commitment

ACM Reference Format:
Chengzhi Lu, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Guodong Yang, and Chengzhong Xu. 2023. Understanding and Optimizing Workloads for Unified Resource Management in Large Cloud Platforms. In *Eighteenth European Conference on Computer Systems (EuroSys '23), May 8–12, 2023, Rome, Italy*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3552326.3587437

1 Introduction

A common approach to achieving high resource utilization in today's large-scale data centers is to co-locate latency-sensitive (LS) applications (such as microservices [35]) and best-effort (BE) applications (such as batch processing jobs [10]) in the same cluster [6, 14, 25, 33, 46, 54, 62]. To facilitate the management of different types of applications, cloud providers such as Google [20], Microsoft [39], and Alibaba [1] have adopted hybrid scheduling over the past decade, implementing separate schedulers for managing BE and LS. For example, Alibaba builds **Sigma** scheduler [8] for scheduling long-running LS while maintaining **Fuxi** scheduler [66] to schedule BE jobs that consist of many small tasks with complex dependencies. By designing customized policies for LS and BE, hybrid scheduling attempts to achieve the best of both worlds in terms of application performance [9, 12, 32].

A fundamental limitation of hybrid scheduling is that one workload scheduler makes seemingly independent scheduling decisions regardless of the co-existence of the other [8], thereby resulting in suboptimal scheduling results. Data center management systems often reserve certain resources for each scheduler, which can easily lead to low overall utilization [4, 9, 24, 56, 58]. To mitigate these limitations, data centers have begun to evolve from hybrid scheduling to unified scheduling through a single scheduling framework, such as Google Borg [56], Alibaba unified resource management system [2, 53], and Facebook Twine [52]. Under this new scheduling paradigm, resource requests from different applications are consistently unified by the system for achieving global scheduling coordination between all applications. At the same time, the unified scheduler can have a complete view of data center resources to improve utilization and scheduling quality.

Although unified scheduling can achieve higher resource utilization, it needs to ensure that the performance of various applications is comparable to that under traditional hybrid scheduling. Under unified scheduling, tasks usually run in containers, which are more susceptible to interference. However, existing unified schedulers do not explicitly quantify task performance [4, 52]. In addition, unified scheduling puts forward high requirements for scheduling efficiency, that is, real-time scheduling of all tasks in a large-scale data center, e.g., Google Borg needs to schedule more than 250K task requests per hour [56]. To design a unified scheduler that satisfies all these requirements, it is critical to deeply understand the characteristics of unified scheduling, but no such analysis has been investigated so far.

Characterizing production workloads. In this paper, we aim to make a comprehensive characterization of unified scheduling in production environments. Specifically, we analyze the workloads of more than one million pods from Alibaba data centers over eight days. We characterize resource usage metrics for tasks (running in containers) from both LS and BE applications. The characterization generates several interesting observations. For example, it shows that unified scheduling incorporates valley filling and peak shaving by carefully scheduling BE applications at appropriate times, which can not only provide performance guarantees for LS workloads, but also improve utilization when the workload of LS is low. The characterization also highlights that, despite the peak-load shifting, the overall resource utilization remains low, i.e., less than 30% on average. At the same time, pods still have long scheduling delays waiting for both CPU and memory resources, and the scheduling delay follows a heavy-tailed distribution. These results indicate that there is a lot of room for improvement in resource utilization under unified scheduling.

Datacenter schedulers often overcommit resources to allow the sum of resource requests on a machine to exceed its capacity, thereby increasing resource efficiency [4]. A key to achieving this is to accurately predict the actual resource usage of each physical machine. We quantify how various existing prediction methods proposed by industry [4] perform on Alibaba production workloads. The quantification results show that these methods can lead to severe overestimation, indicating the need for more resource-efficient prediction approaches in the context of unified scheduling.

We also characterize the performance of pods for BE and LS applications, showing they are highly correlated with pod resource usage and resource contention on the physical host. Furthermore, pods within the same application behave fairly consistently, suggesting the profiles of individual applications can be obtained based on prior history to predict pod performance. Therefore, machine learning algorithms could be used online to predict the behavior of each pod to produce more efficient scheduling results.

Optimizing unified scheduling. Following observations from our characterization, we design Optum, a unified data center scheduler for scheduling unified requests, aiming to improve resource utilization while achieving good performance for each task running in the container. Optum maintains two profiles for each application, i.e., the pod resource usage and pod running performance. The performance profile also captures resource interference.

Moreover, Optum implements a more accurate predictor to predict future resource usage on each physical machine. The principle behind this predictor is that the peak of the total resource usage of any two pods from different applications is far below the sum of the peak usage of these two pods. As such, the predictor combines the estimates of resource usage for all pod pairs to produce more accurate and compact predictions.

Based on application profiles and the resource usage predictor, Optum builds a global optimization framework to maximize the difference between the overall resource utilization in data centers and the performance degradation of all pods caused by resource contention. To tackle this

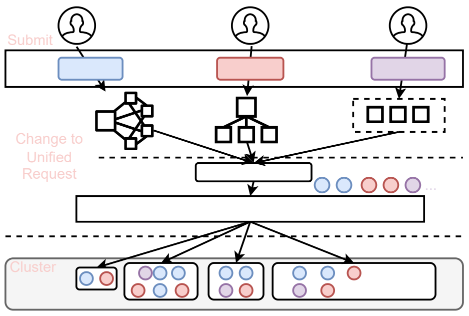


Figure 1. Alibaba unified scheduling framework.

problem, Optum designs scalable solutions that can generate scheduling decisions with a small overhead.

We develop a trace-driven testbed to evaluate Optum. It emulates a data center of about 6000 servers with the same configuration as that in Alibaba data centers, which operate with real-world workloads. The evaluation results show that Optum can improve resource utilization by up to 15% compared to state-of-the-art unified schedulers while achieving the same performance for all pods.

Our contributions. We have made the following contributions in this paper:

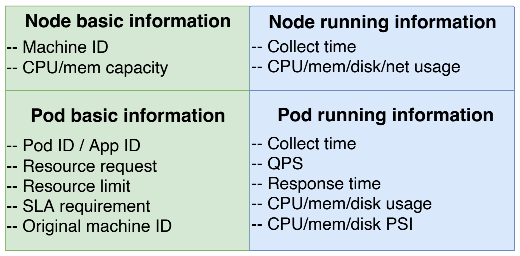
- We conduct a comprehensive study of unified scheduling in large-scale data centers. Our study reveals several interesting observations about resource usage and pod performance. Furthermore, our study highlights the inefficiency of existing resource usage predictors for resource over-commitment.
- We design a new unified data center scheduler that can well balance the trade-off between resource utilization, pod performance, and scheduling scalability.
- We run large-scale experiments driven by production workloads from Alibaba data centers to demonstrate the superiority of our designed scheduler compared to existing solutions.

2 Unified Scheduling Background and Alibaba Trace Overview

In this section, we will provide a brief overview of Alibaba's unified scheduling framework, and introduce Alibaba data center workloads to understand unified scheduling policies as well as the running status of various applications in Alibaba data centers.

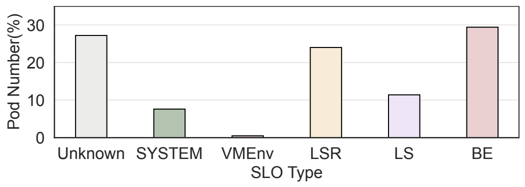
2.1 Alibaba Unified Scheduling Architecture

Unified scheduling in Alibaba data centers has fully unified the scheduling of e-commerce, search and promotion, MaxCompute, and Ant businesses. It can greatly support large-scale resource scheduling in dozens of data centers with millions of containers [2]. Each of Alibaba's data centers is located in a region, and each regional data center is divided into multiple clusters based on its customers and



Node basic information	Node running information
-- Machine ID -- CPU/mem capacity	-- Collect time -- CPU/mem/disk/net usage
Pod basic information	**Pod running information**
-- Pod ID / App ID -- Resource request -- Resource limit -- SLA requirement -- Original machine ID	-- Collect time -- QPS -- Response time -- CPU/mem/disk usage -- CPU/mem/disk PSI

(a) Trace information



(b) Pod SLO distribution

Figure 2. Alibaba trace overview

environment. Each data center or cluster has its own independent unified scheduler.

We show Alibaba's unified scheduling framework in Fig.1. Before an application (such as a MapReduce job or a long-running web service) can run its tasks under unified scheduling, it first follows its application-specific task execution plan, reconstructing a set of task requests according to the Unified Request format, and then submits these requests (with affinity requirements) to the API Server. After receiving the task requests, the unified scheduler selects an appropriate physical host to place each task following its scheduling policy. Specifically, the scheduler first selects the nodes satisfying the affinity as the candidate nodes for the task, and then ranks these candidate nodes according to their balance between the resource utilization and the SLO (Service Level Objective) requirement of the task request. Then an agent located on the selected host will start a pod that consists of several containers to run the new scheduled task.

2.2 Alibaba Trace Overview

In this paper, we analyze the characteristics of over a million pods deployed in one Alibaba data center consisting of two separate clusters, each containing nearly 6000 physical hosts. We collect the running status of all submitted pods as well as all physical hosts over an 8-day period. Compared with hybrid scheduling traces published in 2018 [32], these new traces [1] include more information about pod performance metrics. The trace information is shown in Fig. 2(a), we will give a detailed description of these traces in the following.

Node. Each physical node/host/machine can be identified by a unique machine ID. The CPU and memory capacity of each node is normalized by the tracing system, and their capacities are similar to that in previous traces [32]. The new traces also record the utilization and usage of the CPU,

[1]Traces are available at https://github.com/alibaba/clusterdata

memory, network, and disk of the node at runtime, with a sampling interval of 30 seconds.

Pod. Under unified scheduling, tasks from all applications run in pods. Each pod corresponds to one task, and has one pod ID, one application ID, and one machine ID that identifies the node hosting the pod when the pod is first scheduled. Pods with the same application ID usually provide the same service. The basic information of pods recorded in the traces also contains resource requirements of pods such as resource requests and resource limits. As with Kubernetes, a resource request specifies the number of resources a pod asks to run, while the resource limit denotes the maximum resource usage that the pod is allowed to use.

At the same time, the traces include the collection time of each record and various performance metrics of pods at different levels. Specifically, these metrics include application-level metrics such as the average pod response time and QPS (query per second) over the sampling interval of one minute. As for OS-level metrics, the tracing system records the normalized CPU, memory, and disk usage. Additionally, in order to characterize the interference between different pods on the same physical host, the OS-level metrics also include PSI (Pressure Stall Information), such as CPU, memory, and disk PSI [61]. The sampling interval for OS-level metrics for each pod is 30 seconds. To control overhead, the tracing system does not record hardware-layer metrics for pods.

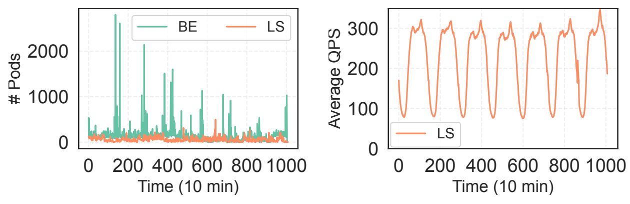
In the current configuration of Alibaba's unified scheduling, there mainly exist three different levels of SLO requirements among all pods, including BE (Best Effort), LS (Latency Sensitive), LSR (Latency Sensitive Reserved). LSR pods are for production services and have stricter performance requirements than LS pods and need to bind a fixed number of CPU cores, while BE is typically used for batch applications and LS is used for long-running services. As shown in Fig. 2(b), except for BE, LS, and LSR, which account for 70% of the total pods, the majority of the remaining pods are UNKNOWN pods that do not provide any class information. There are also SYSTEM and VMEnv pods that have no explicit SLO requirements. Among all these pods in the traces, pods with high-reliability requirements (i.e., LSR + LS) account for over 35%. In this paper, we focus on the characterization of pods with *explicit* SLO requirements.

3 Characterization of Unified Scheduling Workloads

In this section, we conduct an in-depth study of Alibaba's unified scheduling. The study covers resource utilization, pod performance, scheduling policy, and scalability analysis.

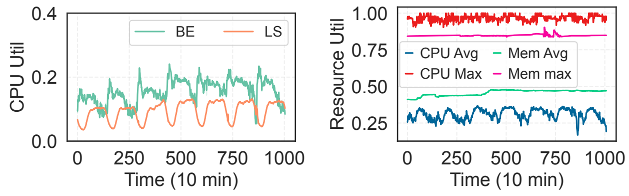
3.1 Resource Utilization Characteristics

3.1.1 Improvement of utilization. Unified scheduling is expected to achieve higher resource utilization than traditional hybrid scheduling. To validate this, we quantify the resource utilization of all physical hosts in two clusters over eight days. To begin with, we first show how the workloads of different types of applications vary over time. Here, we merge the LS and LSR pods into one group because they have similar resource utilization patterns. As shown in Fig. 3(a), the number of BE pods submitted in each interval (of ten minutes) is significantly larger than that of LS and LSR pods, which maintain a fairly constant submission rate. At the same time, the average number of service requests (represented by QPS) sent to each LS pod (LSR pod) exhibits a distinct periodical pattern, which is mainly caused by customers' regular activities [36].



(a) Numbers of submitted pods (b) Average QPS of all LS pods

Figure 3. Workloads over time.



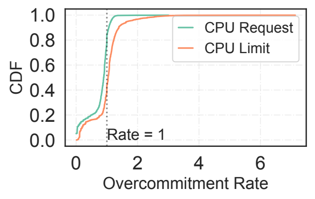
(a) Average Pod CPU utilization (b) Host resource utilization

Figure 4. Resource utilization under unified scheduling.

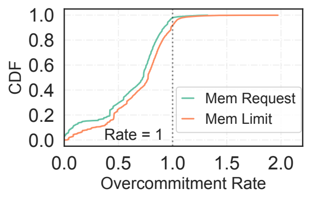
Due to the periodicity of QPS, the CPU utilization of LS pods also varies periodically over days, as shown in Fig. 4(a). More interestingly, Fig. 4(a) also shows that the fluctuation of the utilization of BE pods tends to be opposite to that of LS pods. As a result, unified scheduling achieves higher overall resource utilization than hybrid scheduling without good global coordination between different types of schedulers (i.e., **Fuxi** and **Sigma**). Specifically, as shown in Fig. 4, the maximum CPU utilization across all hosts achieved by unified scheduling can almost always reach 100%, which is much higher than that under hybrid scheduling, i.e., 80% [34]. It also shows that memory utilization exhibits higher stability compared to CPU utilization.

Implication 1: By carefully scheduling BE pods at appropriate times, unified scheduling can fill valleys and shave peaks between the resource usage of all pods.

3.1.2 Resource over-commitment. Although the unified scheduling can improve resource efficiency, the average resource utilization achieved in Alibaba's data centers remains very low, about 30% for CPU and 40% for memory, as shown in Fig. 4(b). Resource utilization also varies greatly across different physical hosts.

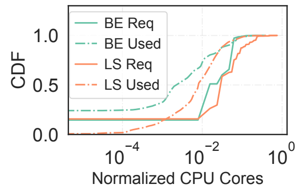


(a) CPU overcommitment rate

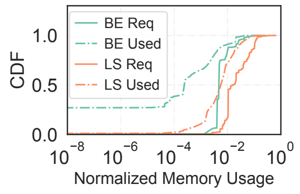


(b) Memory overcommitment rate

Figure 5. The distribution of resource overcommitment rate across all hosts.



(a) CPU request and usage



(b) Memory request and usage

Figure 6. The distribution of (normalized) resource requests and actual usage across all pods.

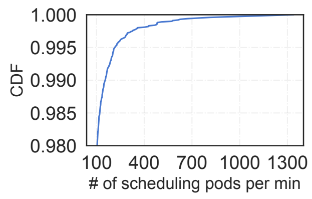


Figure 7. Distribution of the number of pods to be scheduled in each minute.

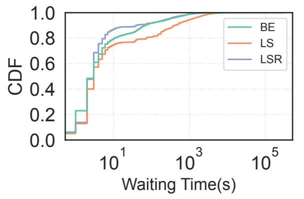
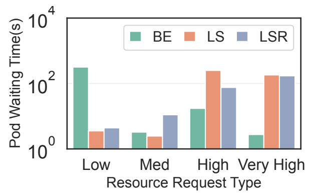
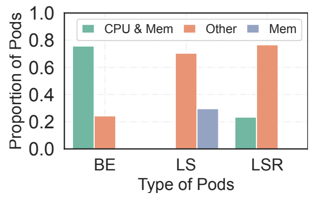


Figure 8. Distribution of waiting time for pods with different SLO types.



(a) Average waiting time



(b) Source of delay

Figure 9. Waiting time of pods and the type of insufficient resources that cause scheduling delays. Other reasons include affinity requirements, insufficient temporary storage space, etc.

This low utilization is even with the adoption of resource over-commitment, which allows the sum of resources allocated to the pods on a host to exceed its physical capacity. As shown in Fig. 5(a), the over-commitment rate of CPU requests can reach as high as four. Here, the over-commitment rate is the ratio of the sum of the resource requests of all pods to the host capacity. The over-commitment rate of the CPU limit is even much higher, i.e., as high as seven. Moreover, on average, a host over-commits CPU resources with a probability higher than 0.25 when it is hosting pods. While for memory, the system conservatively makes over-commitment to avoid running out-of-memory, which can kill all programs on the host. As shown in Fig. 5(b), a host over-commits memory with a probability less than 0.03.

However, the actual resource usage is far below the resource request for both BE pods and LS (LSR) pods. As shown in Fig. 6(a), more than 75% of BE pods use at most 0.01 (normalized) CPU cores during execution, while they request nearly 0.03 CPU cores. For LS pods, this gap is even higher, i.e., 5×. In addition, memory resources are almost fully utilized by BE pods but under-utilized by LS pods, as shown in Fig. 6(b). This results in a large waste of memory resources.

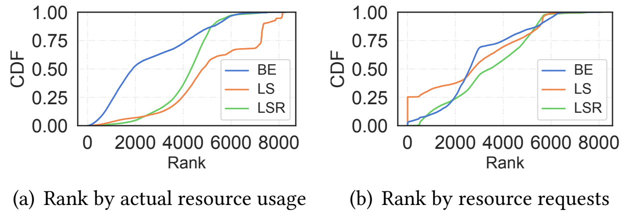
Implication 2: With unified scheduling, resources in Alibaba data centers are over-committed adequately but still significantly underutilized.

3.1.3 Pod scheduling delay. In Alibaba data centers, the number of pods to be scheduled within each minute follows a heavy-tailed distribution. As shown in Fig. 7, the unified scheduler should schedule pods from less than 100 to occasionally over 1000 per minute, thus putting a high requirement on scheduling scalability.

Although the overall resource utilization in the two clusters is low on average, a large number of pods still have long scheduling delays waiting for resources. As shown in Fig. 8, the waiting time of pods also follows a heavy-tailed distribution. In particular, over 10% of BE pods experience a waiting time of more than 100 seconds, which is up to 30× the total running time of BE pods. For LS pods, they suffer from a longer tail than BE pods in terms of the distribution of waiting time though they have higher priority. This can significantly degrade the performance of LS pods especially if they need to scale quickly to handle sudden increases in service workload. The key reason behind the long tail of waiting time is that the unified scheduler adopts a more conservative over-commitment policy for LS pods than for BE pods (the details are presented in § 3.2). In contrast, LSR pods have shorter waiting times than BE pods because the unified scheduler can preempt BE pods for them.

Interestingly, BE pods with smaller resource requests (0.01 to 0.02 CPU cores) wait longer than pods with larger resource requests (more than 0.08 CPU cores) which is against the trends of LS and LSR pods, as shown in Fig. 9(a). This is because the number of BE pods with small resource requests is relatively large, which easily leads to resource contention with other pods. In order to ensure the SLA of LS and LSR pods, the unified scheduler often delays the scheduling of BE pods, resulting in longer waiting times.

Moreover, both CPU and memory could be resource bottlenecks when scheduling pods in Alibaba's data centers. As shown in Fig. 9(b), more than half of BE pods are delayed for scheduling due to insufficient CPU and memory. In contrast, the scheduling delay of LS pods is mainly due to insufficient memory or other reasons such as affinity requirements. While for LSR pods, the scheduling delay is caused by insufficient CPU and memory.



(a) Rank by actual resource usage (b) Rank by resource requests

Figure 10. The CDF of ranks of selected hosts for all pods.

Implication 3: The pod submission rate follows a heavy-tailed distribution. Both CPU and memory can become resource bottlenecks causing scheduling delays for BE and LS pods. Improving resource utilization for CPU and memory in data centers, therefore, becomes critical for mitigating scheduling delays.

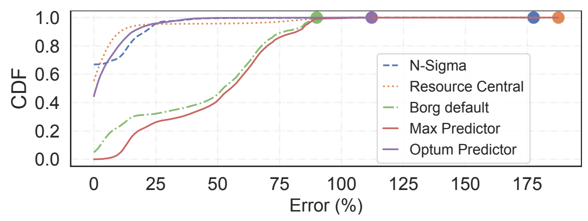
3.2 Resource Over-commitment Policies

Resource over-commitment has been an efficient approach to improving resource utilization in data centers. With resource over-commitment, the scheduler needs to predict the resource usage on all physical hosts [4] and then selects the host with the minimum available resources that can fit the pod for scheduling.

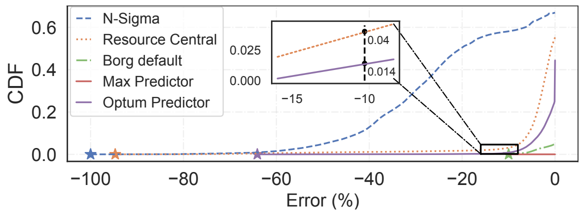
3.2.1 Over-commitment policies for different types of pods. Conservative over-commitment policies can guarantee the running performance of LS (LSR) pods but negatively impact the resource efficiency and pod scheduling delay. By contrast, overly aggressive policies can improve resource utilization with a cost of significantly sacrificing the performance of LS(LSR) pods.

In this part, we evaluate how the scheduling results of Alibaba data centers match the scheduling decisions produced by these two extreme policies. Under this evaluation, aggressive (conservative) policy predicts resource usage on each host as the overall resource usage (resource requests) of all pods in the last scheduling interval. Additionally, to schedule multi-dimensional resources that are represented by a vector, i.e. $\langle \text{CPU}, \text{memory} \rangle$, schedulers in production clusters usually select the host with the largest alignment score [21], where the score is defined as the inner product between the resource request vector of pod p and the resource usage or requests vector of host h.

We compute the rank of the selected host for each pod in Alibaba traces according to the alignment score under two different over-commitment policies. As shown in Fig. 10(a), when using the first policy that adopts the actual resource usage as the predicted outcome, the ranks of the selected hosts for BE pods are pretty high, i.e., more than 60% of selected hosts rank in the top 1/4. In contrast, only 20% of selected hosts rank in the top 1/4 when using the second policy with total resource requests as the predicted outcome, as shown in Fig. 10(b). Furthermore, LS (LSR) pods show a completely opposite trend to BE pods. This indicates that Alibaba unified scheduler tends to over-commit BE pods



(a) The distribution of over-estimation errors



(b) The distribution of under-estimation errors

Figure 11. CPU usage prediction by different approaches. ★ denotes the maximum underestimation error and ● denotes the maximum overestimation error.

based on the actual resource usage but hardly over-commits when scheduling LS Pods. This results in the significant scheduling delay for LS pods as illustrated in Fig. 9(a). In addition, this policy also hurts BE pods due to insufficient resources as future resource usage of LS pods may greatly exceed the resource usage of the last scheduling interval, as shown in Fig.4(a).

Implication 4: When designing unified schedulers in large-scale data centers, it is desirable to implement a more aggressive over-commitment policy on LS pods to reduce their scheduling delays. At the same time, accurate prediction of resource usage is critical to ensuring the running performance of all pods.

3.2.2 The accuracy of different resource usage predictors. In this part, we investigate how various existing resource-usage predictors perform on Alibaba's unified scheduling workloads, which motivates us to design more effective over-commitment policies. We first describe several widely adopted resource prediction methods in industry [4].

Borg Default [3]. This method is the default resource usage prediction method used by Google Borg. It takes the sum of the resource requests of all pods on the machine, multiplied by a fixed ratio λ, as the prediction for future resource usage. $\lambda = 1.0$ reduces to a conservative policy that assumes pods will use all their request resources. And $\lambda = 0.9$ is widely used in many real systems.

Resource Central [9]. Microsoft's Azure clusters use this approach, which adopts the sum of the k-th percentage of resource usage for each pod on the host as a predicted peak in future resource usage. k is usually 99.

N-sigma [4]. This method assumes the overall resource usage on each host follows a Gaussian distribution, which has been validated by many practical systems [26]. Following this assumption, N-sigma computes the mean of $\overline{RU_h}$ as well as the N times standard deviation std(RU_h) of the overall resource usage on a host h over the last period (usually lasts for 24 hours) as the prediction of the resource usage of the host. N is usually 5.

Max Predictor [4]. MaxPredictor takes the maximum resource utilization prediction among the above three predictors as its final prediction.

We evaluate the efficiency of these predictors by comparing the difference between their predicted resource usage and the ground truth using one-day samples as: Error $= (\hat{R}_h^u - R_h^u)/R_h^u$, where $\hat{R}_h^u$ is the predicted resource usage and R_h^u denotes the ground truth.

If Error < 0, it indicates that the predictor under-estimates resources, which can result in performance degradation. Conversely, if Error > 0, it indicates an over-estimation, which can cause resource wastage.

Fig. 11(a) shows that both Borg Default and MaxPredictor can lead to severe over-estimations while N-sigma tends to under-estimate resources as shown in Fig. 11(b). Specifically, with a probability of 0.5, Borg Default can over-estimate resource usage by at least 50%. And using N-sigma may result in underestimating the resource usage by as much as 25%, with a probability of around 0.4. When it comes to the Max Predictor, its prediction results lead to a higher overestimation rate compared to all other predictors. This is because Max Predictor chooses the maximum prediction among all predictors as its final result.

Resource Central and Optum Predictor have comparable performance. They both achieve high prediction accuracy on average by measuring the sum of the resource usage of all pods within the same node. However, Optum Predictor has a key advantage over Resource Central: it further measures the dynamic resource usage of pods accurately, as explained in Section 4.2.2. As a result, the error distribution under Optum Predictor has a much smaller tail than Resource Central. Fig. 11(a) shows Resource Central can overestimate resources by up to 175%, whereas Optum Predictor stays below 110%. Moreover, Resource Central can also underestimate resources significantly in certain cases, with a maximum error of 95%, which is 30% higher than Optum Predictor, as shown in Fig. 11(b). Furthermore, Resource Central is three times more likely than Optum Predictor to under-estimate resource usage by more than 10%, as highlighted in the zoomed-in box of Fig. 11(b).

Implication 5: Existing resource usage predictors can lead to severe over-estimations and thus greatly degrade resource efficiency for unified scheduling workloads. Designing a more accurate predictor is important for improving resource utilization and ensuring pod performance.

3.3 Application Running Performance

When designing aggressive over-commitment policies, it is critical to well control the performance degradation of pods caused by interference. In this section, we investigate how pod performance can be impacted by resource usage and resource contention.

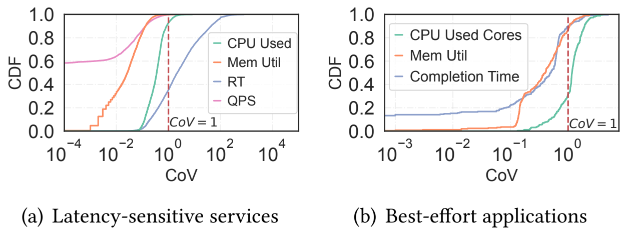


Figure 12. The distribution of the CoV of the behavior of pods within the same application. The behavior includes both resource usage and pod performance.

3.3.1 Behavior of pods within the same application.

First, we explore how the behavior of pods varies from each other in the same application. Specifically, we quantify the coefficient of variation (CoV = standard deviation divided by average) of the average pod resource usage and pod performance for each application. The pod performance here refers to the response time (RT) of LS (LSR) pod and the completion time of BE pod. We also investigate the QPS of LS and LSR pods within each application.

As shown in Fig. 12(a), for most LS applications, the resource usage of pods behaves fairly consistently. In particular, more than 90% of the applications have a CoV below 1. Moreover, the QPS of most applications has a very small CoV (less than 0.1), implying that the QPS in each application is well balanced across all pods. However, the RT of pods shows low consistency. Specifically, only 40% applications have the CoV of RT less than 1. This is because a service request is usually handled by multiple pods in an application [35]; the RT of one pod includes the processing time of all pods it depends on. Therefore, RT is not a good indicator for quantifying the performance of LS pods. BE applications also exhibit consistent behavior in terms of pod completion time and memory utilization, as shown in Fig. 12(b). In contrast, the CPU usage of pods within an application is not as consistent as memory. This is because batch applications are often data-intensive and the input size of each pod can vary widely.

Implication 6: For most applications, their pods behave fairly consistently in terms of resource utilization (such as CPU and memory utilization for LS applications) and running performance (such as Completion Time for BE applications). Therefore, the profile of each individual application can be obtained as a-priori information to help a unified scheduler improve scheduling quality.

3.3.2 The performance of LS pods.

As mentioned above, the RT metric could not be a good indicator of LS applications' performance. Therefore, in this section, we investigate the performance of LS pods in other layers to find a better indicator. Since LSR pods behave similarly to LS pods, in

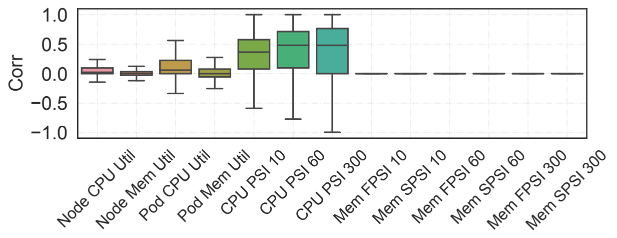


Figure 13. Distribution of the correlation between pod Response Time and OS-level metrics. Here, FPSI means *full PSI* and SPSI means *some PSI*.

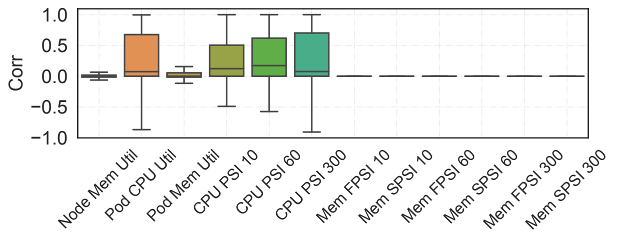


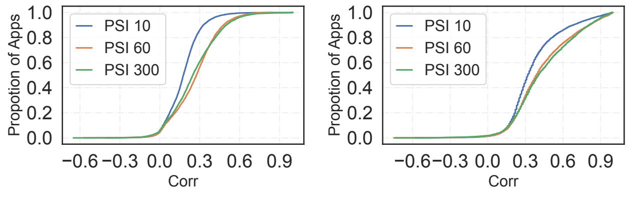
Figure 14. Distribution of the correlation between pod QPS and OS-level metrics.

this part, we do not distinguish between them and treat LSR pods as special LS pods.

In the literature, many works have adopted the hardware-metric CPI (cycles per instruction) as the container performance indicator [32, 63]. However, the cost of collecting this metric for a large number of pods is too high in the industrial environment. Therefore, we resort to the use of OS-level metrics, such as CPU, memory utilization, and PSI. PSI measures the ratio of the average waiting time of processes under a cgroup on the CPU cores or memory they use to the sampling interval [61]. Consequently, PSI quantifies the interference caused by insufficient resources and its impact on complex workloads or even the entire system. The operating system provides three different sampling intervals for PSI, with lengths of 10, 60, and 300 seconds, respectively. For example, CPU *PSI 10* computes the percentage of time in the last 10 seconds that a task is stalled due to CPU contention. It is worth noting that there are two different types of PSI metrics, *full* and *some*, which respectively represent the ratio of the time that all tasks or some tasks wait for resources to the sampling interval. Since it is not possible for all tasks to stall the CPU, only the *some* PSI metric is applicable to the CPU.

As shown in Fig. 13, for more than 50% of applications, the correlation coefficient between pod RT and all types of CPU PSI is greater than 0.5 on average. Compared to PSI, CPU and memory utilization of pod and host show a much weaker correlation with pod RT. At the same time, the memory-related PSIs have little correlation with the pod RT.

In principle, the performance of LS pods is a function of pod resource usage, resource contention, and QPS. To further demonstrate that CPU PSI is a good indicator for capturing pod performance, we quantify the relationship



(a) Host CPU utilization (b) Pod CPU utilization

Figure 15. Distribution of correlation between PSI metrics and different resource usage metrics.

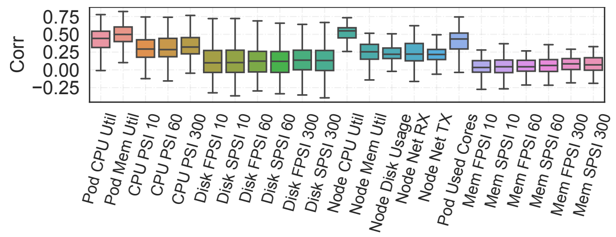


Figure 16. Correlation between BE pod completion time and different resource metrics. In addition to utilization and PSI, we include RX and TX to reflect network conditions. Here RX refers to the total number of bytes received by the pod, and TX refers to the total number of bytes sent by the pod.

between CPU PSI and QPS under fixed pod resource usage and resource contention. As shown in Fig. 14, the PSIs also have a relatively strong correlation with QPS, i.e., more than 50% of applications have a positive correlation between QPS and PSI 60.

Moreover, as shown in Fig. 15(a) and Fig 15(b), in addition to QPS, PSIs (especially *PSI 60*) are strongly positively correlated with host CPU utilization and pod CPU utilization. And interestingly, the correlation between the PSI and host utilization grows with pod utilization. This suggests that we can profile the relationship between these metrics for optimizing unified scheduling.

3.3.3 The performance of BE pods.

Unlike LS pods, for BE pods, application-level metrics such as pod completion time can directly capture their running performance. Therefore, in this part, we study which factors can affect the pod completion time for best-effort applications.

As shown in Fig. 16, the completion time of pods is highly correlated with both pod resource utilization and node resource utilization. Specifically, for over 75% of best-effort applications, the correlation coefficient between pod completion time and node CPU utilization is greater than 0.5. In addition, for over 50% of best-effort applications, the correlation coefficient between pod completion time and node memory utilization is greater than 0.25. These results indicate that resource contention on CPU and memory can severely prolong the completion time of BE pods. Therefore, profiling the relationship between these metrics can help to improve running performance for best-effort applications.

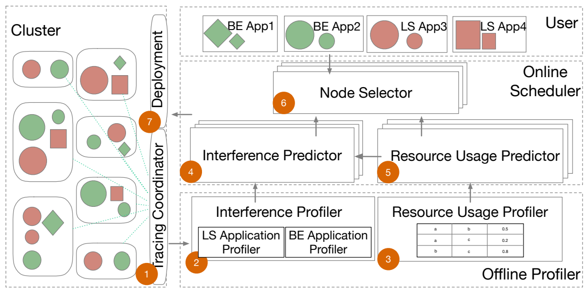


Figure 17. The system architecture of Optum.

Implication 7: The PSI index is a good measure of the performance of LS pods. Additionally, machine learning can be used to profile the PSI for each latency-sensitive application in terms QPS, the pod resource utilization and host resource utilization, and profile pod completion time with respect to the pod and host resource utilization.

3.4 Principles for Optimizing Unified Scheduling

There is a lot of room to increase resource utilization and reduce task scheduling delays under unified scheduling. To fill this room, data center management systems need to implement more accurate resource usage predictors to overcommit resources for both BE and LS pods.

Providing performance guarantees for LS applications is always the first priority of resource management. Fortunately, the relationship between resource utilization and performance for most LS applications can be profiled by applying machine learning methods. Based on these profiles, the resource management system can jointly optimize resource utilization and pod performance.

High scalability is another critical requirement for optimizing unified scheduling considering that the number of submitted pods per minute can be very large. As a consequence, the resource management system can only seek simple yet effective solutions when solving large-scale optimization problems.

4 Design of Optum

Inspired by the characterization study on Alibaba's unified scheduling workloads, we design Optum, a data-center scheduler for optimizing unified resource management while maintaining high scalability. In this section, we present the design details of Optum.

4.1 Overview of Optum System

Optum deploys a *Tracing Coordinator* (❶ in Fig. 17) on top of the resource management system. The *Tracing Coordinator* is responsible for collecting both OS-level and application-level metrics from all pods and physical hosts running in the data center. It stores all tracing data in a centralized database.

Optum includes an *Offline Profiler* that works in the background. *Offline Profiler* consists of two components, *Interference Profiler* (❷ in Fig. 17) and *Resource Usage Profiler* (❸ in Fig. 17). *Interference Profiler* uses the tracing data collected from *Tracing Coordinator* to build a performance profile for each application. This profile models the impact of resource usage and resource contention on pod performance. At the same time, *Resource Usage Profiler* estimates the resource usage of pods from each pair of applications.

The key module of Optum is *Online Scheduler*, which makes a scheduling decision for a newly submitted pod. It consists of three components including *Interference Predictor* (❹ in Fig. 17), *Resource Usage Predictor* (❺ in Fig. 17), and *Node Selector* (❻ in Fig. 17). *Interference Predictor* predicts the impact of resource contention on all pods after the scheduling of a new pod. *Resource Usage Predictor* predicts the overall resource usage when allocating a new pod to this host. *Node Selector* selects a host that can well balance the trade-off between resource utilization and interference.

Based on the decision made by *Online Scheduler*, *Deployment Module* (❼ in Fig. 17) deploys the new pod to the selected host. And if some pods are scheduled to the same host at the same time, the *Deployment Module* also needs to resolve the conflict.

4.2 Offline Profiling

4.2.1 Interference Profiler.

Interference Profiler builds a performance profile for each application individually while their tasks behave fairly consistently. Each profile captures the sensitivity of the pod to resource contention, which helps the scheduler mitigate potential performance degradation when trying to schedule more jobs to the same physical host.

For each LS applications S_i, *Interference Profiler* builds a model to quantify the PSI, which serves as a proxy for application-level performance. The input features of this model include pod CPU and memory utilization C_p^t and M_p^t, host CPU and memory utilization C_h^t and M_h^t, and normalized QPS Q^t. Furthermore, due to the long-running nature of online services, all these inputs features together with the PSI indicator capture timely information. *Interference Profiler* learns a function f_i^o as follows.

$$\mathrm{PSI}_p^t = f_{S_i}^o(\mathrm{C}_p^t, \mathrm{M}_p^t, \mathrm{C}_h^t, \mathrm{M}_h^t, \mathrm{Q}^t). \tag{1}$$

For each BE application B_j, *Interference Profiler* also builds a separate learning model to learn the pod completion time CT_p (normalized to the maximum pod completion time) for each application. Since completion time is a long-term outcome, the input features to this model only contain long-term metrics, including the maximum pod CPU and memory utilization C_p^m and M_p^m, and maximum host CPU and memory utilization C_h^m and M_h^m. The model can be expressed by a function f_j^b as shown below.

$$CT_p = f_{B_j}^b(\mathrm{C}_p^m, \mathrm{M}_p^m, \mathrm{C}_h^m, \mathrm{M}_h^m). \tag{2}$$

In order to improve the accuracy of the model, Optum discretizes the PSI and job completion time and then uses the discretized values as the ground truth of the model. Specifically, Optum divides the space of prediction into multiple buckets, and then takes the upper bound of the bucket as the final prediction. For example, when the PSI is divided into ten buckets and the prediction falls into the 0.2 to 0.3 bucket, the final prediction will be 0.3. By comparing several models, Optum adopts Random Forest as it can yield the highest accuracy among various models.

4.2.2 Resource Usage Profiler.

Resource Usage Profiler is another important component of *Offline Profiler* mainly designed to generate resource usage predictions as close to the ground truth as possible for each host.

The key principle behind the profiler is that the maximum overall resource usage of any two pods over time is much smaller than the sum of the maximum resource usage of the pods. Mathematically, let $\mathrm{C}^u_p(t)$ and $\mathrm{C}^u_q(t)$ denote the CPU usage of pod p and pod q at time t, respectively, we have:

$$\max_t \left\{\mathrm{C}^u_p(t) + \mathrm{C}^u_q(t)\right\} \leq \max_t \left\{\mathrm{C}^u_p(t)\right\} + \max_t \left\{\mathrm{C}^u_q(t)\right\}. \quad (3)$$

Following this principle, *Resource Usage Profiler* computes an effective resource usage coefficient $\mathrm{ERO}(\cdot)$ for each pair of applications. Take applications A and B as an example, *Resource Usage Profiler* records the sum of the resource usage of two pods p and q (p from A and q from B) whenever they are co-located on the same physical host. It then divides this sum by the total resource requests of these two pods as the resource usage ratio, i.e.,

$$\mathrm{RO}_{p,q}(t) = \frac{\mathrm{C}^u_p(t) + \mathrm{C}^u_q(t)}{\mathrm{C}^r_p + \mathrm{C}^r_q} \leq 1, \quad (4)$$

$\mathrm{ERO}(\cdot)$ is defined as the maximum value of all resource usage ratios computed for applications A and B, i.e.,

$$\mathrm{ERO}(A, B) = \max_{p \in A_p, q \in B_p} \max_t \mathrm{RO}_{p,q}(t), \quad (5)$$

where A_p and B_p represent the set of pods from A and B, respectively. The computation of $\max_t \mathrm{RO}_{p,q}(t)$ can be parallelized on each machine, allowing for efficient processing. The *Resource Usage Profiler* then only needs to collect the resulting values from different machines in the cluster and update $\mathrm{ERO}(\cdot)$ accordingly. Here, $\mathrm{ERO}(\cdot)$ is a pairwise metric, which is stored in the cache and updated only when the observed peak resource usage changes. Since a server can only accommodate less than a hundred pods at a time, the update of $\mathrm{RO}_{p,q}(t)$ can be completed within 100μs. So the entire process of updating $\mathrm{ERO}(\cdot)$ for all application pairs can be done within 1 ms. For new applications that have never appeared before, $\mathrm{ERO}(\cdot)$ is initialized to 1.0.

$\mathrm{ERO}(\cdot)$ can also be extended to a tripe-wise metric, under which the profiling of resource usage is performed for each combination of three applications and achieve more precise resource utilization prediction. However, it can incur large profiling overhead. In its current implementation, Optum focuses on pair-wise profiling, which can still significantly increase resource utilization when benchmarked against existing approaches.

Compared to CPU usage, Optum tends to be more conservative when profiling memory usage. For applications whose pods maintain a stable memory utilization (CoV ≤ 0.01), *Resource Usage Profiler* uses the maximum memory utilization of all pods within each application as their profiles. For other applications, *Resource Usage Profiler* profiles their maximum memory utilization as one.

4.3 Online Scheduling

The *Online Scheduler* of Optum aims to balance the trade-off between improving system utilization and ensuring application performance. Before going to the design details of each component, we first introduce the general optimization framework behind the *Online Scheduler*.

4.3.1 Optimization framework.

Consider a set of LS applications I and BE applications J, in order to improve utilization, the scheduling policy should use as few physical hosts as possible to run all submitted pods from I and J, while reducing the performance degradation caused by resource contention. As a result, the *Online Scheduler* aims to maximize a global objective, which is the difference between the host utilization and performance degradation across all pods. Because both CPU and memory can become resource bottlenecks as illustrated in Fig. 9(b), the *Online Scheduler* jointly optimizes CPU and memory utilization. To achieve this, Optum formulates the utilization of each host h as the product of CPU utilization and memory utilization, i.e., $\mathrm{Uti}^C_h \cdot \mathrm{Uti}^M_h$. The performance of pod p from LS application i ($i \in I$) is represented as PSI^p_i, which captures the worst PSI during the execution of p. Similarly, the performance of pod q from BE application j ($j \in J$) is represented as CT^q_j, which describes the normalized completion time of q. The global optimization problem is formulated as:

$$\max \sum_{h \in H} \mathrm{Uti}^C_h \cdot \mathrm{Uti}^M_h - \omega^o \cdot \sum_{p \in P_i, i \in I} \mathrm{PSI}^p_i - \omega^b \cdot \sum_{q \in P_j, j \in J} CT^q_j$$
$$\text{s.t.}, \mathrm{Uti}^C_h,\ \mathrm{Uti}^M_h \leq 1, \forall h \in H, \quad (6)$$

where H denotes the set of non-idle physical hosts in the data center, P_i and P_j represent the set of pods submitted by i and j, respectively. Here, parameters ω^o and ω^b are the weights that characterize the importance of LS applications and BE applications. The design of this objective function enables the resource management system to implement a more aggressive resource over-commitment policy for scheduling LS pods while mitigating their performance degradation. In addition, the system can also impose separate constraints on PSI from important services.

However, globally optimizing this problem is generally difficult as both host utilization and individual pod performance depend on all the pods located on each host (per Eq. (1) and

Eq. (2)). The search space for feasible solutions can be very large when the number of hosts and pods is huge. And more challenging, the pods are usually submitted over time and the scheduling decisions need to be made in a short time.

To address the above challenges, Optum resorts to the use of efficient greedy solutions, which can yield scalable scheduling policies. At a high level, when a new pod is submitted, the *Online Scheduler* evaluates the performance of scheduling it to each candidate host based on the objective function in Eq. (6), and then greedily selects the candidate that yields the best performance to schedule this pod. The performance evaluation involves two components *Resource usage predictor* and *Interference Predictor*. Meanwhile, *Node Selector* is responsible for selecting the best candidate host.

4.3.2 Resource Usage Predictor. This component predicts the resource usage of the candidate host when a new pod is about to be scheduled to the host. Specifically, *Resource Usage Predictor* first estimates the total resource usage of each pair of two pods following the pod scheduling order based on the profiling results obtained from the *Resource Usage Profiler*. Suppose pods $p_1, p_2, \cdots, p_n$ are running on physical host h and p_i is scheduled before p_{i+1}, and p_{n+1} will be scheduled to h. And these pods are submitted by applications $A_1, A_2, \cdots, A_{n+1}$. The estimation of the total CPU usage of pod p_{2i-1} and p_{2i}, $\mathsf{EC}(p_{2i-1}, p_{2i})$ is given by:

$$\mathsf{EC}(p_{2i-1}, p_{2i}) = \mathsf{ERO}(A_{2i-1}, A_{2i}) \cdot (\mathsf{C}^r_{p_{2i-1}} + \mathsf{C}^r_{p_{2i}}), \quad (7)$$

where $\mathsf{C}^r_{p_{2i-1}}$ denotes the CPU request of pod p_{2i-1}. Based on this estimation, *Resource Usage Profiler* then predicts the overall CPU usage on host h, POC_h as follows:

$$\mathsf{POC}_h = \sum_{i=1}^{\lceil n/2 \rceil} \mathsf{EC}(p_{2i-1}, p_{2i}) + ((n+1) \bmod 2) \cdot \mathsf{C}^r_{p_{n+1}}. \quad (8)$$

For the prediction of the memory usage on host h, *Resource Usage Predictor* simply sums up all the estimation of memory usage of each pod, i.e., $\mathsf{POM}_h = \sum_{i=1}^{n+1} \mathsf{EM}(p_i)$ where $\mathsf{EM}(p_i)$ represents the memory estimation of p_i, which is equal to the product of the maximum memory utilization of A_i (obtained from *Resource Usage Profiler*) and the memory request of p_i.

For those applications whose resource profiles change over the lifetime, *Resource Usage Predictor* may lead to overestimation, because the *Resource Usage Profiler* builds profiles based on the observed maximum resource usage in past history. However, this is not a big deal as it only impacts the improvement of utilization under Optum. Nevertheless, Optum can still significantly outperform the baseline schedulers, as we show in § 5.3.

4.3.3 Interference Predictor. The main function of the *Interference Predictor* is to evaluate the overall resource interference caused by the about-to-be-scheduled pod to all pods on the candidate host. For existing LS pod p (from service S_i), the resource interference RI^p_h is quantified by the estimation of PSI provided by *Interference Profiler*, i.e.,

$$\mathsf{RI}^p_h = f^o_{S_i}(\mathsf{C}^m_p, \mathsf{M}^m_p, \mathsf{POC}_h/Cap^C_h, \mathsf{POM}_h/Cap^M_h, \mathsf{Q}^m_{S_i}), \quad (9)$$

where Cap^C_h and Cap^M_h denote the CPU and memory capacity of host h. C^m_p, M^m_p, and $\mathsf{Q}^m_{S_i}$ represent the maximum CPU utilization, maximum memory utilization and maximum QPS of pods from service S_i, respectively. Similarly, for existing BE pod q (from BE application B_j), the resource interference RI^q_h is quantified by the estimation of pod completion time:

$$\mathsf{RI}^q_h = f^b_{B_j}(\mathsf{C}^m_q, \mathsf{M}^m_q, \mathsf{POC}_h/Cap^C_h, \mathsf{POM}_h/Cap^M_h). \quad (10)$$

For the about-to-be-scheduled pod, its own interference is approximated by the prediction of its performance.

4.3.4 Node Selector. The main role of *Node Selector* is to select the most suitable physical host among multiple candidates to schedule a newly submitted pod according to predictions from *Resource Usage Predictor* and *Interference Predictor*. *Node Selector* first checks the (estimated) resource utilization of each candidate host that meets the affinity requirement. If the estimated resource utilization of candidate h is below one, i.e., POC_h and POM_h do not exceed the physical capacity, *Node Selector* computes a score for h based on the following formula.

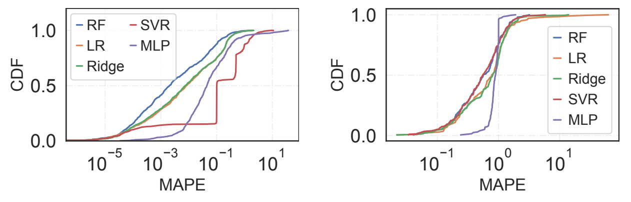
$$\mathsf{Score}_h = \frac{\mathsf{POC}_h}{Cap^C_h} \cdot \frac{\mathsf{POM}_h}{Cap^M_h} - \omega^o \cdot \sum_{p \in L_h} \mathsf{RI}^p_h - \omega^b \cdot \sum_{q \in B_h} \mathsf{RI}^q_h, \quad (11)$$

where L_h and B_h denote the set of LS pods and BE pods that are located on host h (including the newly submitted pod). Finally, the *Node Selector* selects the best candidate that yields the highest score to schedule the new pod.

While the *Node Selector* is able to compute the scores of all nodes in the cluster to select the best one for each pod, this approach can become prohibitively expensive as the cluster size increases. To enhance scheduling scalability, Optum uses the recently developed technique PPO [42], which divides all the physical hosts of the whole data center into multiple groups and scheduling is performed separately within each group. Specifically, Optum randomly selects a set of hosts in the data center as candidates with a fixed sampling probability (0.05) when scheduling a new pod. In the meanwhile, all components of the *Online Scheduler* work in a multi-threaded mode, and each thread is responsible for computing scores for only a few candidate hosts.

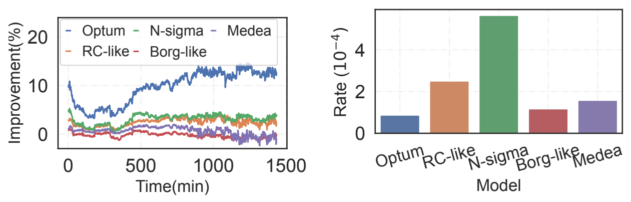
4.4 Deployment Module

The scheduling decisions made by *Online Scheduler* are actually executed by the *Deployment Module*. When the data center scale is very large, the resource management system may include multiple distributed unified schedulers that work in parallel, and each scheduler is responsible for scheduling a portion of submitted pods. As a result, the decisions made by different schedulers can conflict with each other, that is, different pods are simultaneously scheduled to the



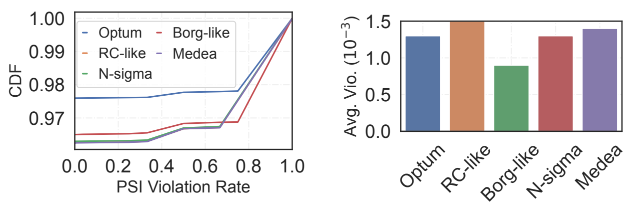
(a) Latency sensitive applications (b) Best-effort applications

Figure 18. Profiling accuracy for LS and BE applications under different learning models.



(a) Resource utilization (b) Violation of resource usage

Figure 19. The improvement of resource utilization and resource usage violation rate under different schedulers compared to the original Alibaba unified scheduler.



(a) PSI violation rate (b) Violation rate of completion time

Figure 20. Performance of pods from latency-sensitive applications and best-effort applications.

same physical host, resulting in inaccurate resource usage predictions and interference predictions. In this case, the *Deployment Module* needs to resolve the scheduling conflict. Specifically, it schedules only the pod that yields the highest score (per Eq. (11)) to the corresponding host and re-dispatches other pods to their *Online Schedulers* for later scheduling.

5 Evaluation of Optum

To evaluate the effectiveness of Optum, we have conducted large-scale trace-driven experiments, and in this section, we present the evaluation results.

5.1 Experiment setup

Experimental Workloads. We implement a trace-driven testbed using workloads from Alibaba data centers. These workloads contain one million pods submitted by 10,000+ applications running on about 6,000 physical hosts over eight days. In addition, the profilers of Optum use the running data of pods in the first seven days to build the learning model.

System Settings. ω^o and $\omega^b$are critical for task scheduling. In the following experiments, we set ω^o and ω^b to 0.7 and 0.3, respectively. Additionally, over-commitment of memory resources may cause too serious consequences such as OOM kill. Thus in this evaluation, to avoid running out of memory, Optum limits the memory utilization of each physical host to 0.8, and once a host reaches this limit, Optum will remove it from the candidate list.

Baseline Schemes. We adopt the following widely used scheduling policies as baselines to compare with Optum in terms of resource utilization and application performance.

Resource-Central [9]. This scheduler is implemented in Microsoft Azure clusters, mainly for scheduling VMs. The scheduler selects the host by checking whether the sum of the 99th percentile CPU usage of all pods located on the host exceeds 0.8 times the host capacity. In addition, the scheduler also limits the over-commitment ratio on each host to 1.2.

N-Sigma [4]. This scheduler assumes that the distribution of the overall CPU usage on each physical host follows a Gaussian distribution, and it adopts the empirical rule to predict resource usage, which is the mean value plus 5 times the standard deviation.

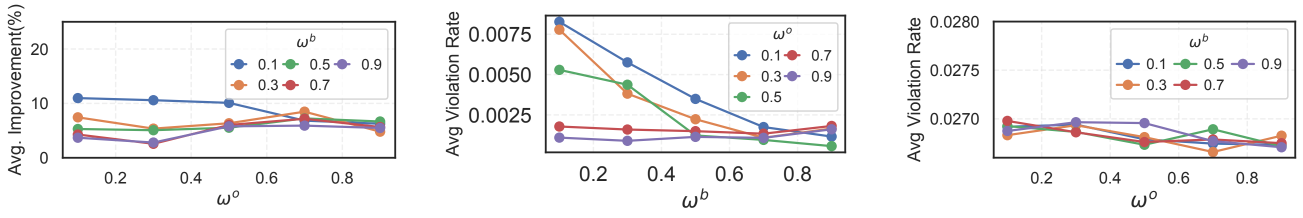
Borg-Like [56]. This scheduler multiplies the total CPU requests of all pods scheduled to the same host by a factor as the prediction of CPU usage, which is widely used in various systems [4].

Medea [17]. This system implements two separate schedulers for long and short-running pods. Medea deploys an Integer Linear Programming (ILP) based scheduler for placing long-running pods which is costly and designs a traditional scheduler for short-running pods with low latency. We set the maximum number of physical hosts to 40 and the maximum number of pods to 15 to find an optimal schedule for long-running pods.

5.2 Accuracy of Interference Predictor

High prediction accuracy is critical for performance optimization under Optum. In this part, we evaluate the prediction accuracy of Optum's *Interference Predictor* using one-day workloads. We discretize the value space of PSI and pod completion time into 25 intervals and map the result to the upper bound within each interval as the predicted value. In addition, we adopt the Mean Absolute Percentage Error (MAPE) as the evaluation metric, which can quantify the distance of predictions relative to the ground truth. We compare the prediction accuracy of Offline Profiler with Linear Regressor (LR), Support Vector Regressor (SVR), Random Forest Regressor (RF), Ridge Regressor (Ridge) and Multi-layer Perceptron Regressor (MLP).

As shown in Fig. 18(a), Random Forest can achieve the highest prediction accuracy. More specifically, for more than 90% of LS applications, the average prediction error of the PSI for pods under the Random Forest model is less than



(a) Average CPU Utilization Improvement (b) BE application performance degradation (c) LS application performance degradation

Figure 21. The sensitivity of Optum to different combinations of parameters.

0.1. In contrast, for nearly 70% of BE applications, the average MAPE of predicting pod completion times is less than 1, as shown in Fig. 18(b). As such, Optum only optimizes the performance for those BE applications (make up 20%) that can achieve high prediction accuracy with MAPE below 0.2. Because LS applications have much higher priority than BE applications in data centers, the learning models built by Optum's interference profiler are accurate enough to mitigate performance degradation caused by resource contention.

5.3 Improvement of Resource Utilization

In this part, we evaluate the end-to-end performance of Optum. We quantify the overall resource utilization achieved by Optum over time and compare it to that achieved by the original Alilbaba's unified scheduler. As shown in Fig. 19(a), Optum can significantly increase the CPU utilization of hosts in the data center compared to Alibaba's scheduler. Specifically, the improvement becomes stable over time, up to 15%. As a comparison, other schedulers can only improve the utilization by nearly 5%. Therefore, Optum can outperform these baselines by 10%. This improvement mainly comes from the accurate prediction of resource usage achieved by Optum. As shown in Fig. 11, the predictor of Optum can predict the resource usage on each host with a lower prediction error than other commonly used predictors.

At the same time, Optum seldom causes resource usage violations, i.e., the overall resource usage on each physical host barely exceeds the host's capacity. As shown in Fig. 19(b), the violation rate under Optum is below 0.01 on average, and all schedulers perform similarly on this metric.

5.4 Evaluation of Running Performance

We proceed to evaluate the performance degradation of pods caused by resource contention. For each pod scheduled to a particular host by a new scheduler, the trace-driven simulator obtains its real performance from the historical traces. More specifically, the simulator collects all the PSI metrics produced by the pod under the same pod utilization, node utilization, and QPS, and takes the maximum PSI as the real performance achieved by the new scheduler. Such performance is compared to the performance of the same pod achieved by Alibaba's unified scheduler.

As shown in Fig. 20(a), for over 97% of LS pods, there is no performance degradation under Optum, that is, the PSI achieved by Optum is no greater than that achieved by Alibaba's scheduler. On average, Optum reduces the violation rate by 1% compared to baseline schemes. Moreover, the PSI of nearly 98% of pods is increased by at most 40% under Optum, which is similar to other schedulers. These results demonstrate that by building a global optimization framework that carefully balances the trade-off between resource utilization and performance degradation, it is possible to improve resource utilization in data centers while providing performance guarantees for most pods.

For all BE pods, we compare their completion times under different schedulers. We quantify the violation rate per application as the percentage of pods that take longer to complete under the new scheduler than Alibaba's scheduler. As shown in Fig. 20(b), Optum yields a violation rate of 0.0013 on average, which is the lowest among all schedulers.

In addition, we also evaluate the scheduling delay of pods due to insufficient resources. It turns out that the scheduling delay is less than 10 seconds for all pods to be scheduled in the data center. In this sense, Optum can always find available resources when scheduling pods, thanks to the adoption of an efficient resource over-commitment scheme.

5.5 Parameter Selection

In this section, we investigate how two parameters ω^o and ω^b affect the scheduling result. When ω^o and ω^b are small, the scheduler tends to schedule pods to hosts with high CPU and memory utilization, resulting in higher overall resource utilization. As shown in Fig. 21(a), Optum improves the average resource utilization by 12% with small ω^o and ω^b. At the same time, this configuration leads to poor performance for a small fraction of LS and BE applications, that is, around 3% of LS applications suffer performance violation in this case, as shown in Fig. 21(c). By contrast, large ω^o and ω^b will only increase the resource utilization by 5% while ensuring that BE and LS pods have much smaller violation rates. To well balance the performance of applications and the improvement of resource utilization, we choose $\omega^o = 0.7$ and $\omega^b = 0.3$, which have the lowest performance degradation for LS applications.

5.6 Scheduling Overhead

In this part, we quantify the scheduling overhead of Optum to study its scalability in large-scale data centers. While Borg-Like's approach of predicting resource usage by simply

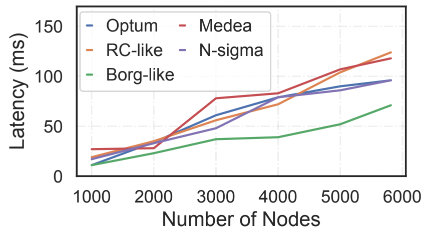


Figure 22. Scheduling overhead under different schedulers.

summing all resource requests results in the lowest overhead, it comes at the cost of significant resource waste. As shown in Fig. 22, the overhead under various schedulers grows almost linearly with the number of servers. Moreover, the average overhead of scheduling a pod across a data center of 6000 servers is 96 ms under Optum, which is lower than other schedulers except for Borg-Like. This means that, Optum can schedule 1000 pods within one minute with only one unified scheduler, which reaches the maximum pod arrival rate in Alibaba data centers. Furthermore, Optum has a maximum scheduling delay of 132 ms, which is also lower than other schedulers.

Optum employs PPO to reduce scheduling overhead, which could potentially compromise scheduling quality. However, pod performance under Optum was not degraded due to our carefully modeled node selection strategy for resource interference. In this sense, Optum can optimize unified scheduling while maintaining high scalability.

6 Related Work

Trace Analysis. In the literature, there exists a bunch of works that focus on the analysis of cloud workloads from different platforms in the context of hybrid scheduling [9, 24, 32, 41, 49, 55, 56]. However, most of these works focus on the study of resource utilization and operational status of clusters or data centers, and lack the characterization of applications running performance and scheduling policies. Other works investigate specific application frameworks including microservices and serverless [35, 38, 51].

Data Center Scheduling. In the past decade, many different types of schedulers have been designed in large-scale data centers to support hybrid workloads. In particular, the centralized schedulers such as YARN-based, have complete knowledge about the entire data center and can fully utilize resources when scheduling applications [5, 9, 13–16, 19, 30, 45, 56–58]. In contrast, two-tier schedulers have better scalability than centralized schedulers by supporting independent schedulers for individual application frameworks [17, 25, 48, 52, 60, 66]. The fully distributed systems adopt multiple schedulers to simultaneously schedule requests from applications where each scheduler is only responsible for a portion of requests [11, 29, 43, 50]. While distributed systems can achieve the highest scalability, they often fail to provide performance guarantees for different types of applications, so do existing unified schedulers [52, 56]. As a comparison, Optum is a new unified scheduler with a globally optimized policy that can well address the trade-off between resource utilization, the performance of different types of applications, and scheduling scalability.

Scheduling Online Services. To handle burst traffic, online applications usually request far more resources than they actually need. Many studies have been conducted to optimize scheduling tailored for online applications. These works mainly utilize machine learning methods or analytical models to identify resource types that are critical for interfering with important services and allocate resources accordingly to ensure their QoS without sacrificing resource efficiency [7, 14, 28, 31, 37, 40, 44, 47, 58, 59, 64, 67]. However, these solutions are not applicable to data center scheduling since they require substantial effort for labeling each service.

Scheduling Batch Applications. There have been designed many efficient scheduling algorithms to reduce the completion time of batch jobs, which are second-class citizens of data centers. In particular, various algorithms quantify the relationship between resource usage and task performance through black-box methods such as Bayesian optimization [27, 46]. Another research direction focuses on optimizing the scheduling order among different tasks by explicitly analyzing the DAG graph [18, 22, 23, 65]. In contrast to these works, Optum focuses on explicitly modeling the impact of resource contention on task completion times for batch applications.

7 Conclusion

In this paper, we conduct a comprehensive analysis of unified scheduling workloads in large-scale data centers. Our study provides several insights into improving resource utilization and optimizing application performance. Motivated by these insights, we design a new unified scheduler Optum. The key novelty behind this scheduler is to implement an accurate resource usage predictor for resource over-commitment and profile task performance for individual applications. Based on this, Optum builds a global optimization framework to maximize resource utilization while ensuring application performance. Moreover, Optum is scalable to make fast scheduling decisions.

Acknowledgments

This work is supported by the National Key R&D Program of China (No.2019YFB2102100), Key-Area Research and Development Program of Guangdong Province (No.2020B010164003), the Science and Technology Development Fund of Macau (0024/2022/A1), and National Natural Science Foundation of China (No. 62072451, 92267105), the Multi-Year Research Grant of University of Macau (MYRG2022-00019-FST), and Alibaba Group through Alibaba Innovative Research Program. We also thank the anonymous reviewers, and our shepherd, Prof. Thaleia Dimitra Doudali, for their helpful comments.

References

[1] Alibaba. 2018. Alibaba Cluster Dataset. https://github.com/alibaba/clusterdata

[2] Aliware. 2021. Unified Scheduling System First Implemented on a Large Scale While Supporting Alibaba's Businesses During Double 11. https://www.alibabacloud.com/blog/

[3] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman, and Nathan DeBardeleben. 2018. On the Diversity of Cluster Workloads and Its Impact on Research Results. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference* (Boston, MA, USA) *(USENIX ATC '18)*. 533–546.

[4] Noman Bashir, Nan Deng, Krzysztof Rzadca, David Irwin, Sree Kodak, and Rohit Jnagal. 2021. Take It to the Limit: Peak Prediction-Driven Resource Overcommitment in Datacenters. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. 556–573. https://doi.org/10.1145/3447786.3456259

[5] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. 2014. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (Broomfield, CO) *(OSDI'14)*. 285–300.

[6] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. 2017. Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. 17–32. https://doi.org/10.1145/3037697.3037700

[7] Shuang Chen, Christina Delimitrou, and José F. Martínez. 2019. PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. 107–120. https://doi.org/10.1145/3297858.3304005

[8] Yue Cheng, Ali Anwar, and Xuejing Duan. 2018. Analyzing Alibaba's Co-located Datacenter Workloads. In *2018 IEEE International Conference on Big Data (Big Data)* (Seattle, WA, USA). 292–297. https://doi.org/10.1109/BigData.2018.8622518

[9] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. 153–167. https://doi.org/10.1145/3132747.3132772

[10] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[11] Pamela Delgado, Diego Didona, Florin Dinu, and Willy Zwaenepoel. 2018. Kairos: Preemptive Data Center Scheduling Without Runtime Estimates. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. 135–148. https://doi.org/10.1145/3267809.3267838

[12] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. 2015. Hawk: Hybrid Datacenter Scheduling. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA) *(USENIX ATC '15)*. 499–510.

[13] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, Texas, USA) *(ASPLOS '13)*. 77–88. https://doi.org/10.1145/2451116.2451125

[14] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. 127–144. https://doi.org/10.1145/2541940.2541941

[15] Christina Delimitrou and Christos Kozyrakis. 2016. HCloud: Resource-Efficient Provisioning in Shared Cloud Systems. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. 473–488. https://doi.org/10.1145/2872362.2872365

[16] Christina Delimitrou, Daniel Sanchez, and Christos Kozyrakis. 2015. Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (Kohala Coast, Hawaii) *(SoCC '15)*. 97–110. https://doi.org/10.1145/2806777.2806779

[17] Panagiotis Garefalakis, Konstantinos Karanasos, Peter Pietzuch, Arun Suresh, and Sriram Rao. 2018. Medea: Scheduling of Long Running Applications in Shared Production Clusters. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Article 4, 13 pages. https://doi.org/10.1145/3190508.3190549

[18] Andrey Goder, Alexey Spiridonov, and Yin Wang. 2015. Bistro: Scheduling Data-Parallel Jobs against Live Production Systems. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA) *(USENIX ATC '15)*. 459–471.

[19] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert N. M. Watson, and Steven Hand. 2016. Firmament: Fast, Centralized Cluster Scheduling at Scale. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. 99–115.

[20] Google. 2020. Google Public Dataset. https://github.com/google/cluster-data

[21] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-Resource Packing for Cluster Schedulers. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. 455–466. https://doi.org/10.1145/2619239.2626334

[22] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. 2016. Altruistic Scheduling in Multi-Resource Clusters. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. 65–80.

[23] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. 2016. Graphene: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. 81–97.

[24] 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 *Proceedings of the International Symposium on Quality of Service* (Phoenix, Arizona) *(IWQoS '19)*. Article 39, 10 pages. https://doi.org/10.1145/3326285.3329074

[25] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA) *(NSDI'11)*. 295–308.

[26] Pawel Janus and Krzysztof Rzadca. 2017. SLO-Aware Colocation of Data Center Tasks Based on Instantaneous Processor Requirements. In *Proceedings of the 2017 Symposium on Cloud Computing* (Santa Clara, California) *(SoCC '17)*. 256–268. https://doi.org/10.1145/3127479.3132244

[27] 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* (Santa Cruz, CA, USA) *(SoCC '19)*. 272–285. https://doi.org/10.1145/3357223.

3362734

[28] Ram Srivatsa Kannan, Lavanya Subramanian, Ashwin Raju, Jeongseob Ahn, Jason Mars, and Lingjia Tang. 2019. GrandSLAm: Guaranteeing SLAs for Jobs in Microservices Execution Frameworks. In *Proceedings of the Fourteenth EuroSys Conference 2019* (Dresden, Germany) *(EuroSys '19)*. Article 34, 16 pages. https://doi.org/10.1145/3302424.3303958

[29] Konstantinos Karanasos, Sriram Rao, Carlo Curino, Chris Douglas, Kishore Chaliparambil, Giovanni Matteo Fumarola, Solom Heddaya, Raghu Ramakrishnan, and Sarvesh Sakalanaga. 2015. Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA) *(USENIX ATC '15)*. 485–497.

[30] Kubernetes. 2022. Kubernetes. https://github.com/kubernetes

[31] Suyi Li, Luping Wang, Wei Wang, Yinghao Yu, and Bo Li. 2021. George: Learning to Place Long-Lived Containers in Large Clusters with Operation Constraints. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. 258–272. https://doi.org/10.1145/3472883.3486971

[32] Qixiao Liu and Zhibin Yu. 2018. The Elasticity and Plasticity in Semi-Containerized Co-Locating Cloud Workload: A View from Alibaba Trace. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. 347–360. https://doi.org/10.1145/3267809.3267830

[33] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '15)*. 450–462. https://doi.org/10.1145/2749469.2749475

[34] Chengzhi Lu, Kejiang Ye, Guoyao Xu, Cheng-Zhong Xu, and Tongxin Bai. 2017. Imbalance in the cloud: An analysis on Alibaba cluster trace. In *2017 IEEE International Conference on Big Data (Big Data)* (Boston, MA, USA). 2884–2892. https://doi.org/10.1109/BigData.2017.8258257

[35] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. 412–426. https://doi.org/10.1145/3472883.3487003

[36] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Jian He, and Chengzhong Xu. 2022. An In-Depth Study of Microservice Call Graph and Runtime Performance. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 3901–3914. https://doi.org/10.1109/TPDS.2022.3174631

[37] Shutian Luo, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Jian He, Guodong Yang, and Chengzhong Xu. 2022. Erms: Efficient Resource Management for Shared Microservices with SLA Guarantees. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Vancouver, BC, Canada) *(ASPLOS 2023)*. 62–77. https://doi.org/10.1145/3567955.3567964

[38] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Eshaan Minocha, Sameh Elnikety, Saurabh Bagchi, and Somali Chaterji. 2022. WISEFUSE: Workload Characterization and DAG Transformation for Serverless Workflows. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 2, Article 26 (Jun 2022). https://doi.org/10.1145/3530892

[39] Microsoft. 2019. Azure Public Dataset. https://github.com/Azure/AzurePublicDataset

[40] Amirhossein Mirhosseini, Sameh Elnikety, and Thomas F. Wenisch. 2021. Parslo: A Gradient Descent-Based Approach for Near-Optimal Partial SLO Allotment in Microservices. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. 442–457. https://doi.org/10.1145/3472883.3486985

[41] Asit K. Mishra, Joseph L. Hellerstein, Walfredo Cirne, and Chita R. Das. 2010. Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters. *SIGMETRICS Perform. Eval. Rev.* 37 (Mar 2010), 34–41. https://doi.org/10.1145/1773394.1773400

[42] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. 2021. Solving Large-Scale Granular Resource Allocation Problems Efficiently with POP. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. 521–537. https://doi.org/10.1145/3477132.3483588

[43] Andrew Newell, Dimitrios Skarlatos, Jingyuan Fan, Pavan Kumar, Maxim Khutornenko, Mayank Pundir, Yirui Zhang, Mingjun Zhang, Yuanlai Liu, Linh Le, Brendon Daugherty, Apurva Samudra, Prashasti Baid, James Kneeland, Igor Kabiljo, Dmitry Shchukin, Andre Rodrigues, Scott Michelson, Ben Christensen, Kaushik Veeraraghavan, and Chunqiang Tang. 2021. RAS: Continuously Optimized Region-Wide Datacenter Resource Allocation. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. 505–520. https://doi.org/10.1145/3477132.3483578

[44] Rajiv Nishtala, Vinicius Petrucci, Paul Carpenter, and Magnus Sjalander. 2020. Twig: Multi-Agent Task Management for Colocated Latency-Critical Cloud Services. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (San Diego, CA, USA,). 167–179. https://doi.org/10.1109/HPCA47549.2020.00023

[45] Jun Woo Park, Alexey Tumanov, Angela Jiang, Michael A. Kozuch, and Gregory R. Ganger. 2018. 3Sigma: Distribution-Based Cluster Scheduling for Runtime Uncertainty. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Article 2, 17 pages. https://doi.org/10.1145/3190508.3190515

[46] Tirthak Patel and Devesh Tiwari. 2020. CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (San Diego, CA, USA). 193–206. https://doi.org/10.1109/HPCA47549.2020.00025

[47] Aidi Pi, Xiaobo Zhou, and Chengzhong Xu. 2022. Holmes: SMT Interference Diagnosis and CPU Scheduling for Job Co-Location. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing* (Minneapolis, MN, USA) *(HPDC '22)*. 110–121. https://doi.org/10.1145/3502181.3531464

[48] Jeff Rasley, Konstantinos Karanasos, Srikanth Kandula, Rodrigo Fonseca, Milan Vojnovic, and Sriram Rao. 2016. Efficient Queue Management for Cluster Scheduling. In *Proceedings of the Eleventh European Conference on Computer Systems* (London, United Kingdom) *(EuroSys '16)*. Article 36, 15 pages. https://doi.org/10.1145/2901318.2901354

[49] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing* (San Jose, California) *(SoCC '12)*. Article 7, 13 pages. https://doi.org/10.1145/2391229.2391236

[50] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. 2013. Omega: Flexible, Scalable Schedulers for Large Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (Prague, Czech Republic) *(EuroSys '13)*. 351–364. https://doi.org/10.1145/2465351.2465386

[51] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference* (Virtual Event, USA) *(USENIX ATC'20)*. Article 14, 205–218 pages.

[52] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, Ben Christensen, Alex Gartrell, Maxim Khutornenko, Sachin Kulkarni, Marcin Pawlowski, Tuomas Pelkonen, Andre Rodrigues, Rounak Tibrewal, Vaishnavi Venkatesan, and Peter Zhang. 2020. Twine: A Unified Cluster Management System

for Shared Infrastructure. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation* (Virtual Event, USA) *(OSDI'20)*. Article 45, 787–803 pages.

[53] Huang Tao and Wang Menghai. 2021. Unveiling Alibaba's Hybrid Scheduling Technology for Complex Task Resources. https://www.alibabacloud.com/blog

[54] Prashanth Thinakaran, Jashwant Raj Gunasekaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. 2017. Phoenix: A Constraint-Aware Scheduler for Heterogeneous Datacenters. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (Atlanta, GA, USA). 977–987. https://doi.org/10.1109/ICDCS.2017.262

[55] Huangshi Tian, Yunchuan Zheng, and Wei Wang. 2019. Characterizing and Synthesizing Task Dependencies of Data-Parallel Jobs in Alibaba Cloud. In *Proceedings of the ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) *(SoCC '19)*. 139–151. https://doi.org/10.1145/3357223.3362710

[56] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. 2020. Borg: The next Generation. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Article 30, 14 pages. https://doi.org/10.1145/3342195.3387517

[57] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing* (Santa Clara, California) *(SoCC '13)*. Article 5, 16 pages. https://doi.org/10.1145/2523616.2523633

[58] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-Scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems* (Bordeaux, France) *(EuroSys '15)*. Article 18, 17 pages. https://doi.org/10.1145/2741948.2741964

[59] Luping Wang, Qizhen Weng, Wei Wang, Chen Chen, and Bo Li. 2020. Metis: Learning to Schedule Long-Running Applications in Shared Container Clusters at Scale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. Article 68, 17 pages.

[60] Yuzhao Wang, Lele Li, You Wu, Junqing Yu, Zhibin Yu, and Xuehai Qian. 2019. TPShare: A Time-Space Sharing Scheduling Abstraction for Shared Cloud via Vertical Labels. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. 499–512. https://doi.org/10.1145/3307650.3326634

[61] Johannes Weiner. 2018. PSI - Pressure Stall Information. https://docs.kernel.org/accounting/psi.html

[62] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. 2013. Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (Tel-Aviv, Israel) *(ISCA '13)*. 607–618. https://doi.org/10.1145/2485922.2485974

[63] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. 2013. CPI2: CPU Performance Isolation for Shared Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (Prague, Czech Republic) *(EuroSys '13)*. 379–391. https://doi.org/10.1145/2465351.2465388

[64] Yanqi Zhang, Weizhe Hua, Zhuangzhuang Zhou, G. Edward Suh, and Christina Delimitrou. 2021. Sinan: ML-Based and QoS-Aware Resource Management for Cloud Microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. 167–181. https://doi.org/10.1145/3445814.3446693

[65] Yunqi Zhang, George Prekas, Giovanni Matteo Fumarola, Marcus Fontoura, Íñigo Goiri, and Ricardo Bianchini. 2016. History-Based Harvesting of Spare Cycles and Storage in Large-Scale Datacenters. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. 755–770.

[66] Zhuo Zhang, Chao Li, Yangyu Tao, Renyu Yang, Hong Tang, and Jie Xu. 2014. Fuxi: A Fault-Tolerant Resource Management and Job Scheduling System at Internet Scale. *Proc. VLDB Endow.* 7, 13, 1393–1404. https://doi.org/10.14778/2733004.2733012

[67] Laiping Zhao, Yanan Yang, Kaixuan Zhang, Xiaobo Zhou, Tie Qiu, Keqiu Li, and Yungang Bao. 2020. Rhythm: Component-Distinguishable Workload Deployment in Datacenters. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Article 19, 17 pages. https://doi.org/10.1145/3342195.3387534