Since January 2020 Elsevier has created a COVID-19 resource centre with free information in English and Mandarin on the novel coronavirus COVID-19. The COVID-19 resource centre is hosted on Elsevier Connect, the company’s public news and information website.

Elsevier hereby grants permission to make all its COVID-19-related research that is available on the COVID-19 resource centre - including this research content - immediately available in PubMed Central and other publicly funded repositories, such as the WHO COVID database with rights for unrestricted research re-use and analyses in any form or by any means with acknowledgement of the original source. These permissions are granted for free by Elsevier for as long as the COVID-19 resource centre remains active.
Improving the cost efficiency of large-scale cloud systems running hybrid workloads - A case study of Alibaba cluster traces

Brad Everman, Narmadha Rajendran, Xiaomin Li, Ziliang Zong *

Texas State University, San Marcos, TX, United States

ARTICLE INFO

Keywords:
Data center efficiency
Server utilization
Workload scheduling
Big data analysis
Alibaba cluster trace

ABSTRACT

The pandemic of coronavirus has dramatically disrupted the retail industry, as many stores are forced to close and people across the world are shelter-in-place with online shopping as the inevitable choice. To meet the rapidly increasing demand for e-commerce, more data centers are expected to provide new or significantly improve existing cloud services that can better support hybrid workloads (e.g. online purchase jobs and batch jobs that support ranking or recommendation systems). Successful cloud systems need to efficiently handle and quickly respond to huge volume of traffic with such hybrid workloads. Meanwhile, it is critical to reduce the total cost of ownership (TCO) for profitability. Improving system utilization is one of the effective techniques to achieve the twin goals of high performance and low TCO. This paper conducts a comprehensive analysis on the 2017 and 2018 cluster traces released by Alibaba, which provides a case study about Alibaba’s best practices in improving the performance and cost efficiency of its large-scale cloud systems by consolidating time-sensitive online service jobs with time-insensitive batch jobs. Our investigation indicates that the over-subscription (causing resource waste and low utilization) and under-subscription (causing performance degradation) problems co-exist in the current Alibaba system. We develop a simulator that allows us to evaluate possible solutions to address this problem and their impact on the performance, energy consumption, and TCO. Our experiments show that the estimated TCO can be reduced by $600,000 for the 2018 trace running on over 4,000 machines without compromising performance. The TCO can decrease by nearly $68 million if similar strategy is extrapolated to Alibaba’s 432,000 web facing servers.

1. Introduction

In the past decade, we have witnessed the rapid growth of e-commerce and cloud services, which created IT giants like Amazon and Alibaba. According to the Netcraft report [22], Alibaba has become the world’s 2nd largest cloud computing company, right next to Amazon. The data centers of Alibaba serve millions of users and process billions of transactions on a daily basis. For example, in the 2017 Single’s Day Shopping Festival (11/11), $25 billion of products were sold and 1.5 billion payment transactions were processed [25] within 24 hours on the Alibaba system. This sales record was five times of the total 2017 Black Friday sales in U.S. [19]. In 2019, the Single’s Day Shopping record had grown to $38.4 billion, a 1.5× increase in two years [13]. The recent coronavirus pandemic has dramatically disrupted the retail industry, which brings a new boom to e-commerce. Leading companies like Amazon and Alibaba have seen flooded online orders. Walmart’s online sales have surged by 74% during the pandemic [30]. Target’s digital sales grew by more than 100% in March 2020 and are up 275% in April, 2020 [28]. It can be projected that more cloud systems will be built to meet the booming demand of e-commerce.

Large-scale cloud systems that can handle such an excessive amount of online traffic are essential to e-commerce, but their enormous appetite for investment and energy are astonishing as well. One of the Alibaba’s newest data centers in Hebei requires ¥18 billion capital investment plus millions of dollars annual electricity bills [31,3]. This number could further grow as the number of servers, the consumed energy, the labor to manage the system, and the electricity price all increase. McKinsey reported that the cost of data centers accounts for approximately 25% of the total corporate IT budget [12]. Inefficient data centers may threaten profitability, despite the large number of users and dominating market share [12]. One of the primary reasons for data center inefficiency is the low utilization. The Gartner and McKinsey report indicated that the server utilization rate is merely 6–12% [12] [23] for most enterprise data centers and another study showed that the...
utilization of Amazon AWS servers is not high either (~7–17%) [15].

To address the inefficiency issue, virtualization and server consolidation have been widely used to boost utilization rate. However, co-running jobs, especially time-sensitive jobs, on the same server may interfere with each other and adversely affect performance. A purchase request failure (even delay) can be disastrous to user experiences and result in huge revenue loss. Alibaba has explored an innovative approach to alleviate this dilemma by scheduling time-sensitive online service jobs and time-insensitive batch jobs to the same machine [33, 34]. In 2017 and 2018, Alibaba released two cluster traces that used the mixed scheduling strategy. The 2017 trace consists of 12,932 batch jobs and 11,076 online service jobs running on 1,313 machines over a 12-hour period. The 2018 trace is at a larger scale with 4,201,015 batch jobs and 370,540 online service jobs running on 4,023 machines over an 8-day period [1].

In this paper, we conduct a comprehensive analysis on the 2017 and 2018 Alibaba cluster traces and make the following contributions:

1. We use the Alibaba trace to quantitatively show that consolidating hybrid workloads can considerably improve the utilization of large-scale cloud systems and reduce TCO. Our analysis on the 2018 trace shows that the servers that only run batch jobs (time-insensitive) or service jobs (time-sensitive) have an average CPU utilization of 29.29% and 7.4% respectively. By applying the mixed scheduling strategy, the average server utilization is improved to 39.26%. This leads to millions of dollars of TCO reduction purely from the reduced number of servers without compromising quality of service (QoS). The TCO benefit will be even larger if savings on the physical footprint, power supplies, maintenance personnel, and cooling facilities are included.

2. We discover that the jobs submitted to the Alibaba cluster request an unreasonable amount of resources. A large portion of jobs (both batch and service jobs) aggressively over-subscribe resources. Meanwhile, a considerable number of jobs (both batch and service jobs) under-subscribe resources. This hurts the Alibaba system in both ways. The under-subscribed jobs will suffer from performance loss while the over-subscribed jobs will waste energy and lower system utilization.

3. We develop a simulator that allows us to quantitatively study the impact of reduced cluster sizes on the performance, energy consumption, and TCO of the Alibaba system.

4. Our simulation results indicate that 5% of servers can be safely removed from the cluster with negligible influence on performance, which is able to save about $600,000 (including server purchase cost and one year of electricity cost) for the system running the 2018 trace. Nearly $68 million TCO reduction can be achieved if similar strategy is extrapolated to Alibaba’s 432,000 Internet-facing servers [22].

The rest of the paper is organized as follows. Section 2 provides background information about the Alibaba system and highlights the trade-offs between server utilization and quality of service (i.e., performance). Section 3 discusses the details of the Alibaba cluster traces, presents the workload analysis results, and describes the identified over-subscription and under-subscription problems in the hybrid workloads. Section 4 presents the design and development of the Alibaba cluster trace simulator. Section 5 illustrates the experimental results conducted using the simulator. The related work is presented in Section 6. Section 7 concludes this study and discusses future work.

2. Background

2.1. Utilization matters

Large scale data centers contain hundreds of thousands of machines, which run a variety of jobs. Most of these jobs cannot fully utilize the CPU and memory resources of the servers thereby causing low server utilization. In fact, this is the primary reason for data center inefficiency because given the same workload, lower utilization means a company has to purchase more servers. AMD reported that the growth in server power use worldwide comes mostly from having more servers, not from having higher power use per server [24]. For a large-scale system, a small increase in server utilization could lead to significant cost savings. Assuming that a data center has a total of 2000 servers (each server costs $3000), among which half of them run batch jobs with an average utilization of 25% and the other half run service jobs with an average utilization of 15%. If the average utilization of servers can be improved to 40%, the system size can decrease by 50% (i.e. remove 1000 servers), which will reduce the TCO by at least $3 million provided that the total workload remains unchanged. Therefore, improving server utilization is an effective way to reduce data center TCO.

2.2. Performance dominates

Server utilization can be improved by consolidating multiple jobs on the same server via virtual machine or container technology. However, this may lead to resource contention and cause unpredictable performance degradation. For the time-sensitive jobs (e.g., online purchasing), request failures or delays are not acceptable because they can result in huge revenue loss. For example, Google found that increasing the search results from 10 per page to 30 per page would increase the page load time from 0.4 to 0.9 s [5]. This half a second performance degradation resulted in a 20% decrease in searches [7], which would cause millions of dollars of loss in advertisement. It is essential that improving server utilization should not be at the cost of sacrificing performance or QoS.

2.3. Alibaba scheduler

As one of the biggest e-commerce providers, Alibaba runs hybrid workloads such as online service jobs (e.g., searching, online shopping, and advertising) and batch jobs (e.g., data mining and machine learning to understand customer behaviors and provide customized recommendations). Since online service jobs are essential for business, they are very time-sensitive. Batch jobs are not time-sensitive so they are given lower priority and will be terminated when resource contention with time-sensitive jobs occurs. To ensure high server utilization and quality of critical online services, Alibaba implemented a mix scheduling framework (see Fig. 1), in which the time-insensitive batch jobs are colocated with the time-sensitive service jobs. More specifically, there are two schedulers in the Alibaba system. The Sigma scheduler allocates online service jobs while the Fuxi scheduler is responsible for dispatching batch jobs [33]. They share the state of the entire cluster to obtain global view of system status and make better scheduling decisions.

3. Workload analysis

3.1. Alibaba dataset

The Alibaba cluster traces provide detailed information about the servers, batch jobs, and service jobs [1]. Specifically, the 2018 trace constitutes 6 files with more than 450 GB of uncompressed data logged over an 8 day period. Each trace file is an SQL dump of its respective table, saved in the CSV format. Since the 9th day dataset has incomplete records (only several hours of data is available and some information is missing), we decide to extract datasets with 7 complete days (i.e. 168 h) for use in our analysis. Fig. 2 visualizes the overall CPU utilization for each day included in the 2018 trace during a 24 h period, from which we can see that the valleys and peaks occur at specific times of the day and the daily usage patterns are largely repetitive.

The server information is given in the machine_meta table and the machine_usage table. The machine_meta table describes the events that
occur on the physical systems such as capacity and errors. The machine_usage table contains the utilization of CPU, memory, and disk access timestamped in 10 second intervals. The batch_task table and the batch_instance table provide information about batch jobs. The batch_task table shows the status and resource demands of the batch tasks. The batch_instance table includes the resource usage of each instance of the task and their status. Service jobs are described in the container_meta table and the container_usage table. Each batch job has one or numerous tasks and each task runs a single or multiple instances, which execute the same binaries but on different input. Each time an instance tries to run is called a trial. An instance can have more than one trial before its successful termination although this is rare (e.g. only less than 1% of instances experience more than one trial).

Table 1 summarizes the total number of servers, batch jobs, tasks, instances, and service jobs in the trace files of the 2017 and 2018 datasets.

|                     | 2017   | 2018   |
|---------------------|--------|--------|
| Number of servers   | 1,313  | 4,023  |
| Batch jobs          | 12,932 | 4,201,014 |
| Batch tasks         | 80,386 | 410,134 |
| Batch instances     | 11,829,897 | 1,351,255,775 |
| Service jobs        | 11,076 | 370,540 |

3.2. Machine utilization

There are a total of 4,023 machines in the 2018 Alibaba cluster trace,
among which 145 machines only run service jobs and 11 machines only run batch jobs. The remaining machines run both service jobs and batch jobs. Figs. 3–5 plot the CPU, memory, and disk utilization of the entire cluster over time in hours, color-coded to indicate machines running service jobs only, batch jobs only, and both jobs respectively. Machines included in machine_usage with no recorded jobs are also displayed for completeness. Table 2 summarizes the average CPU, memory, and disk utilization of the three different kinds of servers as compared to the 2017 trace files. It can be observed that the mixed scheduling strategy considerably improves both CPU and memory utilization. For example, in the 2018 Alibaba trace, the servers that only run batch jobs or service jobs have an average CPU utilization of 29.29% and 7.42% respectively. By applying the mixed scheduling strategy, the average server utilization is improved to 39.26%. We also notice that machines running only service jobs rarely reach 15% CPU utilization while almost all machines running both batch jobs and service jobs can exceed 80% of CPU utilization multiple times. Similarly, the memory utilization of machines running only service jobs is roughly 80% while it is not uncommon for machines running both jobs to exceed 90% of memory utilization. Disk utilization seems to have little impact on performance with less than 8% of disk resources are used on average for mixed systems.

3.3. Batch jobs

When a batch job arrives, the FuxiMaster starts an AppMaster (a.k.a. Application master or job master) for that job. Once started, the AppMaster reads the job description file, interprets the resource requirement, analyzes task dependency, and sends resource requests to the FuxiMaster accordingly. The FuxiMaster allocates resources and sends them to the AppMaster. If there are not enough resources available, it sends the resources in the incremental fashion upon availability. The approved resources sent to the AppMaster is referred to as resource grants. Once the minimum required amount of resources is obtained, the job master starts the task. Resource subscription is the amount of resources that a task requests from the FuxiMaster and gets grants for it. Over-subscription refers to the case in which the requested resources are more than the utilized resources. Similarly, under-subscription refers to the case in which the requested resources are less than the utilized resources. Proper subscription means that the requested resources is roughly equivalent to the utilized resources.

Fig. 6 shows that in the 2017 Alibaba trace, 65.30% of batch jobs over-subscribe CPU resources, among which 13.80%, 23.71%, 30.63% and 31.85% over-subscribe the CPU resources by 0–25%, 25–50%, 50–75%, and 75–100% respectively. These over-subscribed resources will be held by the tasks (without being used) until they terminate, which leads to low utilization and degrades system performance. The under-subscription problem is also common and severe. Our analysis indicates that 19.68% of batch jobs under-subscribe CPU resources, among which 32.68% under-subscribe CPU resources by 25–50% while the other 65.83% under-subscribe CPU resources by over 50%. These jobs could suffer from longer execution time.

3.4. Service jobs

Service jobs are time-sensitive therefore they are given higher privileges in obtaining system resources. To prevent interference from other co-running jobs, service jobs are executed in containers. Each container is guaranteed to have certain amount of CPU, memory and disk space, which is exclusively allocated for it. However, the allocated resources for containers are always not fully utilized. Fig. 7 shows that in the 2017 Alibaba trace, 90.12% of the containers use less than 25% allocated CPU and 8.94% of containers use only 25–50% of CPU resources. As for memory and disk resources, 13.92% of the containers use only less than half of the allocated memory and 96.99% of containers use less than 25% of the allocated disk space.

3.5. Summary

To summarize, the over-subscription and under-subscription problems co-exist in the Alibaba system and are equally severe, which hurts the system efficiency in two ways. The excessive over-subscription of resources causes low resource utilization and decreases the system efficiency. Meanwhile, jobs that under-subscribe resources suffer from delayed execution time due to the lack of sufficient resources.

4. Simulator design

In the previous section, we have analyzed the unique workload characteristics and resource utilization of the Alibaba system. Our analysis indicates that there is a great potential to further improve system efficiency. For example, the system utilization and performance could be improved if the over-subscribed resources are allocated to the
under-subscribed jobs. The number of machines in the cluster could be reduced without degrading performance. Unfortunately, the Alibaba trace was given as static trace files and all parameters in the trace files have been pre-determined. In order to conduct experiments to quantitatively study the impact of varied cluster size and different system configurations on the performance and TCO of the Alibaba system, it is essential to design and develop a simulator that can reproduce the scheduling process. In 2013, Google developed an “Omega” simulator [8] to emulate its cluster scheduling process and explored the impact of different configurations and scheduling strategies on system efficiency and utilization [26]. We adopt the similar methodology used by Google to design and develop our simulator.

In this section, we present the design of the Alibaba scheduler simulator driven by real Alibaba production workloads [33] (it can also take synthetic workloads). The design and implementation are largely

![Average Memory Utilization](image1)

**Fig. 4.** Memory utilization by hours.

![Average Disk Space Utilization](image2)

**Fig. 5.** Disk utilization by hours.

| Year | Type          | CPU Utilization | Memory Utilization | Disk Utilization |
|------|---------------|-----------------|--------------------|------------------|
|      | Batch jobs    | 17.56%          | 29.54%             | 43.31%           |
|      | Service jobs  | 11.27%          | 34.3%              | 35.73%           |
|      | Both          | 28.11%          | 52.18%             | 47.81%           |
| 2018 | Batch jobs    | 29.29%          | 79.55%             | 10.30%           |
|      | Service jobs  | 7.42%           | 80.46%             | 1.94%            |
|      | Both          | 39.26%          | 88.56%             | 7.96%            |

**Table 2**

Average CPU, memory, and disk utilization.

![Table 2](image3)
derived from the details explained in the Alibaba Fuxi paper [34]. The Google Omega scheduler simulator [8] also provides important insights (e.g. how to use Agenda to handle job submission and how to update the event time using a clock) on how to implement a scheduler simulator for large-scale systems. The simulator supports both the 2017 and 2018 Alibaba traces.

4.1. Fuxi scheduler workflow

The Fuxi scheduler includes five main components: the FuxiMaster (FM), the FuxiAgent (FA), the AppMaster (AM), the Task Master (TM), and the Task Worker (TW). Fig. 8 plots the typical workflow that the Fuxi scheduler allocates resources for jobs. Fig. 9 shows the steps to deallocate resources and terminate a batch job. When a job is submitted to the FuxiMaster, the FuxiMaster will first try to find a FuxiAgent residing on a machine with available resources. Next, the FuxiAgent will start the AppMaster for the job. When receiving the job description, the AppMaster first predicts the resource needs for each task then sends resource request to the FuxiMaster. The FuxiMaster will check the available resource pool for free resources, allocate the resources when available and send resource grants to the AppMaster. If there are not enough resources available, the FuxiMaster will push the unfulfilled resource requests into a queue called locality tree and send resource grants in an incremental fashion. Every time when certain resources are granted, the FuxiMaster will update the status of the resource requests. Once the AppMaster receives resource grants, it determines which task to be scheduled next based on task dependency and the resources received. The AppMaster then starts the Task Master for that task and send resources to it. After that, the Task Master will be able to schedule the instances of the task for execution. While a task is running, the Task Worker will periodically report the status of the task to the Task Master. When all Task Workers have reported completion to the Task Master, the Task Master will consider the job to be completed. The Task Master will send resource deallocation request to the AppMaster, which will be forwarded to the FuxiMaster. It is the responsibility of the FuxiMaster to finally deallocate the resources and update the free resource pool. Once all tasks in a job have completed, the AppMaster will report its completion to the FuxiMaster then terminate the job.

4.2. Terminology

To better understand the workflow of the Fuxi scheduler simulator, it is beneficial to briefly introduce four important terminologies: job description, resource requests, resource grants, and locality tree. More details of these terminologies can be found in the Alibaba Fuxi paper [34].

1. Job description – A job description contains job id, job submission time, number of tasks, and task description of each task in the job. Task description includes the task’s creation time, dependency, and
the required resources. In our simulator, a job description is generated for each job residing in the input trace files.

2. Resource request – After receiving the job description, the AppMaster calculates the resource needs of the job and sends resource request to the FuxiMaster. A resource request consists of schedule unit definition, quantities for each schedule unit, and other attributes such as location preference and priority etc. [34], which is sent from the AppMaster to the FuxiMaster to apply for resources.

3. Resource grants – A resource grant is sent by the FuxiMaster to the AppMaster, which carries information about the machine id, the CPU id, memory, and diskspace allocated for the job.

4. Locality tree – In the Fuxi’s scheduler, a job only needs to specify its resource demand once. The FuxiMaster makes incremental resource allocation if the requested resources cannot be fulfilled at once. This can significantly reduce the communication and message processing overhead by preventing jobs from repetitively asserting full resource demands and checking the status of their requests. The FuxiMaster keeps the unfulfilled demands of each job in a locality tree, which supports the incremental resource allocation protocol [34]. In our simulator, a locality tree is implemented to keep unfulfilled resource requests in the waiting queue and automatically grants resources to the AppMaster when new resources are available. Once the requested resources are fulfilled, the FuxiMaster removes the request from the queue. The Alibaba Fuxi paper [34] indicates that jobs with higher priority will get the requested resources early. When the priority is the same, the waiting time will be taken into consideration. Our simulator uses the First In First Out (FIFO) algorithm to resolve the conflict when the priority of multiple jobs is identical.

5. Free resource pool – Upon receipt of resource requests from the AppMaster, the FuxiMaster will check the free resource pool and try to find sufficient free resources which can meet the application’s locality requirements. If the free resource is insufficient, the resource requests will be queued by the FuxiMaster in the locality tree [34]. Before making final allocation decisions, the FuxiMaster considers load balancing of each machine as well. If the load on a machine is higher than the specified load limit, resources on that machine will not be removed from the free resource pool.

4.3. Job submission

The most difficult part is to accurately simulate the submission of jobs to the Fuxi scheduler at the specific time given in the Alibaba trace files or synthetically generated input files. Inspired by the implementation of the Google Omega scheduler simulator [8], we use a priority queue of Agenda that is sorted by time to simulate the job submission process (see Fig. 10). Basically, the Agenda is an event queue that collects the events and sorts them by time. Each event has an action and a specific time at which the action is performed. The simulator maintains a clock that is updated each time an event occurs with the time of that event. Initially, the Agenda is filled with job arrival events and each job arrival event will add new events to the Agenda.
Three input files are generated from the released Alibaba trace files: the batch job file, the batch task file, and the service job file.

1. The batch job file contains the following information: job arrival time, job id, and the number of tasks in the job. The job arrival time is the earliest creation time of a task among all tasks in the job. The number of tasks in the job is counted directly from the corresponding trace file. The job submission time is one second less than the job arrival time, which simulates the time needed to start the AppMaster before the task can be created. In real systems, the task creation time may vary depending on the delay in scheduling the job. Here the one second is a rough estimate because the real delay is not provided in the Alibaba trace files.

2. The batch task file provides information that includes job id, task id, task creation time, the number of instances in the task, duration of each instance, and the amount of CPU, memory and disk space needed for an instance to run. Duration of the instance is calculated from the trace file as the average difference between instance creation time and end time of the successful trial of all instances. The number of instances of the task is calculated by counting final trial of instances with the same job id and task id. The required CPU and memory resources are calculated as the average CPU and memory usage of all instances of the task.

3. The service job file consists of container id, creation time of the container, duration of the container, machine id on which the container is created, the number of CPUs it needs, and their CPU ids as well as the allocated memory and disk space information. All information is available from the container event table except for the duration. As no information exists in the trace files indicating how long a container will exist, we assume the container exists until the end of simulation. We can infer from the trace file that resources for each container are available when they are assigned to a server, thus reallocating container resources as needed.

4.4. Simulator output

The simulator collects and records the following information of each individual machine in a 5-second interval: (1) the amount of CPU, memory, and disk space used; (2) the wait time in queue of each job (time between the job creation and an AppMaster is created for the job); (3) the wait time of each resource request (time between it is created and fulfilled completely); (4) the wait time of each task before it gets first resource grant; and (5) the wait time of each task before it gets last resource grant. The savings on total cost of ownership (TCO) is not directly produced by the simulator. Rather, it is calculated from the simulator output results indirectly after considering the cost of each server, the average power usage of a server, and the electricity price.

4.5. Simulator output

The simulator collects and records the following information of each individual machine in a 5-second interval: (1) the amount of CPU, memory, and disk space used; (2) the wait time in queue of each job (time between the job creation and an AppMaster is created for the job); (3) the wait time of each resource request (time between it is created and fulfilled completely); (4) the wait time of each task before it gets first resource grant; and (5) the wait time of each task before it gets last resource grant. The savings on total cost of ownership (TCO) is not directly produced by the simulator. Rather, it is calculated from the simulator output results indirectly after considering the cost of each server, the average power usage of a server, and the electricity price.

4.6. Limitations

Although we strive to emulate the Alibaba scheduling process as accurately as possible, it is worth noting that our simulator is not able to completely emulate the process due to the lack of panoramic information. Additionally, the simulation may not be 100% accurate due to the following constraints:

1. Referring to Table 1 again, it is obvious the enormity of the 2018 dataset makes it difficult to parse fully in a timely manner. For example, performing the simulation until the cluster has completed all tasks requires an excessive amount of disk space and several days of run-time, making it difficult to perform tests in an efficient manner. Thus, due to memory and time constraints, only the 2017 trace and 24 hours of the 2018 trace are used in the simulation. Since the 2017 and 2018 cluster traces exhibit very similar over-subscription and under-subscription problems and the daily usage patterns in the 2018 traces are highly repetitive (ref. Figure 2), it is reasonable to extrapolate the simulation results to the entire 2018 dataset.

2. The machine utilization information is only available after the first batch jobs are submitted. This could cause some abnormalities due to the lack of a complete snapshot of the cluster. Further, we only consider average machine utilization for jobs as this is all that is provided in the real trace files, i.e. the simulator tries to duplicate the results found in the trace. This could possibly affect the scheduler’s ability to perform most efficiently.

3. Without overhead information of creating the AppMaster, the Task Master, and the Task Worker, the simulator estimates the information, which might affect the waiting time of a job or task and indirectly affects system utilization.

4. The real Alibaba system uses Fuxi scheduler to dispatch batch jobs and the Sigma scheduler to dispatch service jobs. The Fuxi scheduler and Siganma scheduler work together for the mix scheduling policy. Unfortunately, the details about the Sigma scheduler implementation is not published. Our simulator makes simple assumptions that the Sigma scheduler creates a container using the resources mentioned in the trace file without much overhead. In addition, since the trace files do not provide information on how long a container exists, we let the container (once created) exist in the entire life cycle of the simulation. Recall that containers usually request more resources, this might negatively affect the system utilization.

5. The real Fuxi scheduler supports other important functionalities such as fault tolerant scheduling and multi-level black-listing scheme [34], which are not included in our simulator but may affect system utilization.

5. Experimental results

With the simulator, we are able to reproduce the scheduling process and quantitatively evaluate the impact of varied cluster size on system performance and TCO. Specifically, we conduct two groups of experiments and analyze the results. Section 5.1 evaluates the impact of reducing cluster size on system performance, while Section 5.2 analyzes reduction to overall TCO.

5.1. Impact of cluster size

The total number of machines in the 2017 Alibaba trace file is 1,313. Theoretically, reducing the number of machines could possibly increase system utilization. However, this may hurt performance by increasing the wait time of tasks. The question is, to what extent can the cluster size be reduced with a negligible influence on task wait time? This group of experiments is designed to study the impact of varied cluster size on system performance. We reduce the number of machines in the cluster incrementally, observing wait time for resource allocation. Figs. 11 and
show the percentage of tasks that must wait for incremental resource grants. As we can see for the cluster size of 1,313 and 1,248 machines, more than 99% of the tasks are scheduled with no wait time and receive the requested resource as a single grant. However, when the cluster size is reduced further to 1,138 machines, the percentage of tasks that do not wait drops to approximately 96.5%. Similarly, the percentage of tasks that receive all resources as a single grant drops to 95.16%. Fig. 13 shows the number of tasks that have to wait until receiving the first resource grant after sending its resource request. The task is scheduled to start once the first grant is received, so wait time until first grant also means wait time until the task is scheduled. The number of tasks that wait less than 10 seconds until first grant is received for cluster size of 1,313 and 1,138, the latter has a higher wait time by 23 s. But in case of cluster with 1,248 machines, the number of tasks that wait for more than 10 s for the first grant is only 9.

Fig. 14 shows that no task need to wait for more than 10 s for the last grant for a cluster size of 1,313 machines. For the cluster size of 1,248, the number of tasks with wait time until last grant less than 10 s has slightly increased when comparing it with cluster size of 1,313 machines. However, the number of tasks that must wait less than 10 s has significantly increased (close to an unacceptable level) when the cluster size is reduced to 1,138 machines. Preliminary results with the 2018 trace demonstrates comparable response times with a 5% reduction in cluster size. Fig. 15 shows the wait time in seconds until first grant and last grant when simulating the first 24 h of the 2018 dataset for 4,023 and 3,832 machines, respectively. It can be seen that the difference in response time of the cluster is negligible, and further testing on the entire 7 day period is expected to produce results similar to the 2017 trace (see Fig. 2 for the repeated daily patterns).

5.2. TCO reduction

The previous results have shown that the cluster size can be reduced by 5% without degrading performance. In this subsection, we calculate how much TCO reduction can be achieved. According to [34], the Alibaba server contains a 6-core Xeon E5-2430 processor (2.20 GHz), a 96 GB memory and a 12*2T disk array. Machines are connected via two gigabit Ethernet ports. We use the Dell Power Edge T340 server, which has very similar configurations as the Alibaba server specifications, to estimate the TCO. The Dell Power Edge T340 server consumes approximately 1.653 kilowatt-hours (kWh) of power per day and costs $3,078 [4]. We believe this is an extremely conservative estimate as a typical enterprise server purchased in 2020 would cost roughly twice this amount and use at least double amount of energy every day. In our cost calculation, we assume the electricity price of the Alibaba datacenter is $0.11 per kilowatt hour, which is the average data center electricity price reported by Data Center Frontier [21]. We do not consider power savings techniques such as DVFS as power usage information is unavailable in the trace files. We address this concern by using reasonable but conservative estimates of overall system power usage.

Table 3 compares the energy consumption of the two clusters and first resource grant after sending its resource request. The task is scheduled to start once the first grant is received, so wait time until first grant also means wait time until the task is scheduled. The number of tasks that wait less than 10 seconds until first grant is received for cluster size of 1,313 and 1,138, the latter has a higher wait time by 23 s. But in case of cluster with 1,248 machines, the number of tasks that wait for more than 10 s for the first grant is only 9.

Fig. 14 shows that no task need to wait for more than 10 s for the last grant for a cluster size of 1,313 machines. For the cluster size of 1,248, the number of tasks with wait time until last grant less than 10 s has slightly increased when comparing it with cluster size of 1,313 machines. However, the number of tasks that must wait less than 10 s has significantly increased (close to an unacceptable level) when the cluster size is reduced to 1,138 machines. Preliminary results with the 2018 trace demonstrates comparable response times with a 5% reduction in cluster size. Fig. 15 shows the wait time in seconds until first grant and last grant when simulating the first 24 h of the 2018 dataset for 4,023 and 3,832 machines, respectively. It can be seen that the difference in response time of the cluster is negligible, and further testing on the entire 7 day period is expected to produce results similar to the 2017 trace (see Fig. 2 for the repeated daily patterns).

5.2. TCO reduction

The previous results have shown that the cluster size can be reduced by 5% without degrading performance. In this subsection, we calculate how much TCO reduction can be achieved. According to [34], the Alibaba server contains a 6-core Xeon E5-2430 processor (2.20 GHz), a 96 GB memory and a 12*2T disk array. Machines are connected via two gigabit Ethernet ports. We use the Dell Power Edge T340 server, which has very similar configurations as the Alibaba server specifications, to estimate the TCO. The Dell Power Edge T340 server consumes approximately 1.653 kilowatt-hours (kWh) of power per day and costs $3,078 [4]. We believe this is an extremely conservative estimate as a typical enterprise server purchased in 2020 would cost roughly twice this amount and use at least double amount of energy every day. In our cost calculation, we assume the electricity price of the Alibaba datacenter is $0.11 per kilowatt hour, which is the average data center electricity price reported by Data Center Frontier [21]. We do not consider power savings techniques such as DVFS as power usage information is unavailable in the trace files. We address this concern by using reasonable but conservative estimates of overall system power usage.

Table 3 compares the energy consumption of the two clusters and first resource grant after sending its resource request. The task is scheduled to start once the first grant is received, so wait time until first grant also means wait time until the task is scheduled. The number of tasks that wait less than 10 seconds until first grant is received for cluster size of 1,313 and 1,138, the latter has a higher wait time by 23 s. But in case of cluster with 1,248 machines, the number of tasks that wait for more than 10 s for the first grant is only 9.

Fig. 14 shows that no task need to wait for more than 10 s for the last grant for a cluster size of 1,313 machines. For the cluster size of 1,248, the number of tasks with wait time until last grant less than 10 s has slightly increased when comparing it with cluster size of 1,313 machines. However, the number of tasks that must wait less than 10 s has significantly increased (close to an unacceptable level) when the cluster size is reduced to 1,138 machines. Preliminary results with the 2018 trace demonstrates comparable response times with a 5% reduction in cluster size. Fig. 15 shows the wait time in seconds until first grant and last grant when simulating the first 24 h of the 2018 dataset for 4,023 and 3,832 machines, respectively. It can be seen that the difference in response time of the cluster is negligible, and further testing on the entire 7 day period is expected to produce results similar to the 2017 trace (see Fig. 2 for the repeated daily patterns).
Extrapolated TCO comparison.

Table 5 compares the TCO. By shrinking the 2018 cluster size from 4,023 to 3,832 (95% of its original size), we can reduce the total cost of ownership by $600,574. It is worth noting that this number only includes the reduced hardware purchase cost and power savings on the server side. The real TCO benefit will be much larger if the reduced cost on server maintenance and power savings on the cooling facilities are also included.

A cursory analysis of the TCO in Table 4 reveals substantial savings, albeit on a small scale. Alibaba’s large size, however, makes this almost insignificant when considering their $12 billion yearly operating costs as of 2018 [20]. Per Netcraft [22], Alibaba had 432,000 Internet-facing servers as of 2017, and this number is most likely much larger today. Table 5 calculates the TCO reduction if 5% of Alibaba’s 432,000 Internet facing servers can be removed (i.e. removing 21,600 servers in total). The results show that TCO will reduce almost $68 million overall, which includes nearly $66.5 million of hardware capital savings and over $1.4 million annual electricity bill conservation. The TCO benefit will be even larger if savings on physical footprint of data centers, maintenance personnel, and cooling facilities are also considered. Meanwhile, less energy consumption allows more rapid growth due to the ability to use existing power grids and infrastructure, thus increasing overall long-term profitability.

6. Related work

Alibaba workload analysis. Since Alibaba released its cluster trace in 2017, a number of papers have been published to analyze the Alibaba workloads. Lu et al. identified several imbalance scenarios (e.g. spatial imbalance, temporal imbalance, imbalanced CPU and memory utilization per workload, and imbalanced resource demands from different tasks) in the 2017 Alibaba trace [17]. Liu et al. analyzed the Alibaba trace from the system elasticity and plasticity perspectives [16]. They reported that most containers exhibited fairly steady disk and memory usage but the CPU utilization varied. They also identified that batch instances were over-committed for both CPU and memory resources. Cheng et al. [2] and Jiang et al. [10] analyzed the workload characteristics of the 2017 Alibaba dataset with similar conclusions that container resource usage was over-provisioned and thus batch jobs must use whatever free resources were available when run. These conclusions were borne out by the analysis done by Lu et al. [18], which further demonstrated that system resources were not allocated in a balanced way for the heterogeneous workloads. Beyond workload analysis, Li and Hu took one step further and proposed a deep reinforcement learning based job scheduling (DeepJS) algorithm that used the Alibaba trace as a training set [14]. They claimed that DeepJS was able to make better decisions about scheduling than the default heuristic-based approaches, thus performing more efficiently. Tian et al. examined the dependency structures of Alibaba jobs and compared the task dependency of Alibaba jobs with other benchmarks such as TPC-DS and TPC-H [27]. Guo et al. studied the resource efficiency issue of the Alibaba system by analyzing the 2018 dataset [11]. They discovered that memory became the new bottleneck, batch jobs were treated as second-class citizens, and the Java Virtual Machine (JVM) caused the resource inefficiency of containers running time-sensitive service jobs. Unfortunately, despite various problems being identified, most of the existing analysis work can only provide suggestions but were not able to tackle the discovered problems with verified solutions. This is largely because the practical schedulers deployed on large systems are very complicated and the data provided in the trace files are static. To evaluate the effectiveness of possible solutions, it is essential to develop an accurate simulator. For example, Google developed the Omega scheduler simulator to emulate its cluster scheduling process and explored the impact of different configurations and scheduling strategies on system efficiency and utilization [26].

Cloud system scheduler simulator. There are very few published works discussing how to design and develop simulators for large-scale and complex systems. The Mesos scheduler was first introduced by Hindman et al. for sharing commodity clusters between multiple diverse cluster computing frameworks such as Hadoop and MPI [9]. It has been widely adopted by the Hadoop community after it became one of the Apache’s open source projects [6]. The Mesos scheduler enabled fine-grained sharing of resources across frameworks and provided strong support for running distributed tests at scale. The other Hadoop scheduler YARN [29], which has a master daemon that communicates with the clients and one or multiple worker daemons that launch and track processes spawned on a machine. A recent study even enhanced YARN for better heterogeneous resource allocation [32]. The Omega scheduler [26] was specifically designed to address the low utilization and efficiency problem at Google’s production system by allocating and running a mix of workloads (CPU intensive, memory intensive, batch jobs and low-latency jobs) on the same cluster. To some extent, the Alibaba’s mixing schedulers (Fuxi [34] and Sigma) adopted the design philosophies of the Omega scheduler, which leveraged time-insensitive jobs for higher resource utilization. To the best of our knowledge, our simulator is the first attempt to emulate the Alibaba scheduling process and our analysis is the first one to evaluate the Alibaba system from both the performance and cost efficiency perspectives.
7. Conclusions and future work

E-commerce has experienced rapid growth in the past decade and embraced a new boom during the coronavirus pandemic. As more vendors consider online sales as essential business, the number of data centers that support cloud services with hybrid workloads will rise quickly. To make a profit, these data centers must have exceptional performance with reasonable TCO. This paper discusses the best practices of Alibaba by analyzing its 2017 and 2018 cluster traces. Our analysis confirmed that consolidating the time-sensitive service jobs with time-insensitive batch jobs can significantly improve system utilization and reduce TCO. Meanwhile, we discovered that the over-subscription and under-subscription problems co-exist in the Alibaba system. Addressing this problem will have great potential to further reduce TCO without compromising performance. In addition, a simulator is developed to reproduce the scheduling process using the Alibaba trace, which allows us to evaluate the impact of varied cluster size on the performance (in term of task wait time) and TCO of the Alibaba system. Our experimental results indicate that reducing the size of the cluster running the Alibaba 2018 trace by 5% can save approximately $600,000 of TCO without degrading system performance. The total TCO reduction will scale up to nearly $68 million if extrapolated to Alibaba’s 432,000 Internet facing servers.

This work can be further extended to reduce the resources allocated for containers and allocate them for under-subscribed batch jobs, provided that the performance of time-sensitive service jobs will not be affected. Moreover, we can further improve the simulator by addressing some of the limitations stated in Section 4.6.

Conflict of interest

None declared.

Declaration of Competing Interest

The authors report no declarations of interest.

References

[1] Alibaba, Alibaba Cluster Trace Program, 2017. https://github.com/alibaba/clusterdata.
[2] Y. Cheng, Z. Chai, A. Anwar, Characterizing Co-located Datacenter Workloads: An Alibaba Case Study, 2018.
[3] ChinaDaily, Ali Green Intelligent Data Center Settled in Zhangbei, Will Become the Heart of Northern Data, 2016. http://www.chinadaily.com.cn/interface/yid/1120781/2016-09-12/c_d/26769440.html.
[4] Dell, Poweredge t340 Tower Server, 2020. https://www.dell.com/en/business/p/poweredge-t340-pd.
[5] D. Farber, Google’s Marissa Mayer: Speed Wins, 2006. https://www.zdnet.com/article/googles-marissa-mayer-speed-wins/.
[6] A.S. Foundation, Mesos Architecture, 2012. http://mesos.apache.org/documen-tation/latest/architecture/.
[7] Google, Search Algorithms, 2012. https://www.youtube.com/watch?v=Qow-AkDrk8.
[8] Google, Cluster Scheduler Simulator Overview, 2013. https://github.com/google/cluster-scheduler-simulator.
[9] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker, I. Stoica, Mesos: a platform for fine-grained resource sharing in the data center, Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, USA (2011) 295–308.
[10] C. Jiang, G. Han, J. Lin, G. Jia, W. Shi, J. Wan, Characteristics of co-allocated online services and batch jobs in internet data centers: a case study from Alibaba cloud, IEEE Access 7 (2019) 22495–22508.
[11] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, Y. Bao, Who limits the resource efficiency of my datacenter: an analysis of Alibaba datacenter traces, Proceedings of the IEEE/ACM International Symposium on Quality of Service (IWQoS 19) (2019).
[12] J. Kaplan, W. Forrest, N. Kindler, Revolutionizing Data Center Energy Efficiency, 2006. https://safian.org/pdf-docs/Mckinsey_Data_Center_Efficiency.pdf.
[13] S. Klebnikov, Alibaba’s 11/11 Singles’ Day by the Numbers: A Record 38 Billion Haul, 2019. https://www.forbes.com/sites/sergeiklebnikov/2019/11/1/1alibaba-as-1111-singles-day-by-the-numbers-a-record-38-billion-haul-9e6bb02c7272.
[14] F. Li, B. Hu, Deepie: Job Scheduling Based On Deep Reinforcement Learning in Cloud Data Center, 2019, pp. 48–53, https://doi.org/10.1145/3354843.3355513.
[15] H. Liu, A measurement study of server utilization in public clouds, Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, IEEE Computer Society, USA, 2011, pp. 435–442, https://doi.org/10.1109/DASC.2011.87.
[16] Q. Liu, Z. Yu, The Elasticity and Plasticity in Semi-Containerized Co-locating Cloud Workload: A View from Alibaba Trace, 2018, pp. 347–360, https://doi.org/10.1109/BigData.2017.8258257.
[17] C. Lu, K. Ye, G. Xu, C.Z. Xu, T. Bai, Imbalance in the Cloud: An Analysis on Alibaba Cluster Trace, 2017, pp. 2884–2892, https://doi.org/10.1109/BigData.2017.8258257.
[18] C. Lu, K. Ye, G. Xu, C.Z. Xu, T. Bai, Imbalance in the Cloud: An Analysis on Alibaba Cluster Trace, 2017, pp. 2884–2892, https://doi.org/10.1109/BigData.2017.8258257.
[19] I. Linden, Black Friday Racks Up $5.03b on Mobile Alone, 2017. https://techcrunch.com/2017/11/24/black-friday-deals-net-640m-in-sales-so-far-mobile-60-of-all-tra-fic/.
[20] MarketRealist, Alibaba Group Holding Ltd, 2020. https://marketrealist.com/tic/km.v/.
[21] R. Miller, The Cloud Becomes A Major Force in Green Energy, 2017. https://datanet-terfrontier.com/the-cloud-becomes-a-force-in-green-energy/.
[22] Netcraft, Cloud Wars: Alibaba Becomes 2nd Largest Hosting Company, 2017. https://www.netcraft.com/archives/2017/08/22/cloud-wars-alibaba-becomes-2nd-largest-hosting-company.html.
[23] J. Ouyang, J.R. Lange, H. Zheng, Shoosh4tc: using vmm assists to optimize tib operations on preempted vcpu, SIGPLAN Not. 51 (2016) 17–23, https://doi.org/10.1145/3007611.2892248.
[24] P. Robichaux, Calculating Server Power Costs, 2008. http://www.itprovided.com/m/windows-78/calculating-server-power-costs.
[25] J. Russell, Alibaba Smashes Its Single’s Day Record Once Again As Sales Cross $25 Billion, 2017. https://techcrunch.com/2017/11/11/alibaba-smashes-its-singles-da-y-record/.
[26] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, J. Wilkes, Omega: flexible, scalable schedulers for large computer clusters, Proceedings of the 8th ACM European Conference on Computer Systems, Association for Computing Machinery, New York, NY, USA (2013) 351–364, https://doi.org/10.1145/2465351.2465386.
[27] H. Tian, Y. Zheng, W. Wang, Characterizing and synthesizing task dependencies of data-parallel jobs in Alibaba cloud, Proceedings of the ACM Symposium on Cloud Computing (SoCC 19) (2019).
[28] K. Tyko, Target Digital Sales Make Significant Gains Because of Covid-19 Demand, But In-Store Sales Drop, Driving Shares Down, 2020. https://www.usatoday. com/story/money/2020/04/23/coronavirus-pandemic-target-online-growth-due -covid-19/3007311001/.
[29] V.K. Vavilapalli, A.C. Murdoch, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Carino, O. M’alley, S. Radiana, B. Adhikari, E. Baldeschwieler, Hadoop hadoop: yet another resource negotiator, Proceedings of the 4th Annual Symposium on Cloud Computing, Association for Computing Machinery, New York, NY, USA (2013) 289–290, https://doi.org/10.1145/2526316.2526333.
[30] P. Wahba, Walmart’s Online Sales Surge During the Pandemic, 2020. https://fo rum.com/2020/05/19/walmart-online-sales-amazon-e-commerce/.
[31] B. Youngag, How to Reduce the Cost of Cloud Computing Data Center?, 2016. https://www.technopreneur.com/news/20161110/HowToReduceCloudCostV7S1.html.
[32] X. Zhang, Y. Liu, Y. Wu, C. Zhao, Mibache: a global scheduler for mixed workloads in heterogeneous environments, J. Parallel Distrib. Comput. 111 (2018) 93–103, https://doi.org/10.1016/j.jpdc.2017.07.007.
[33] Z. Zhang, About Alibaba Cluster and Why We Open the Data, 2017. https://github.com/alibaba/clusterdata/wiki/About-Alibaba-cluster-and-why-we-open-the-data.
[34] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, J. Xu, Fuxi: a fault-tolerant resource management and job scheduling system at internet scale, Proc. VLDB Endow. 7 (2014) 1393–1404, https://doi.org/10.14778/2733004.2733012.