ABSTRACT

Cloud platform provides great flexibility and cost-efficiency for end-users and cloud operators. However, low resource utilization in modern datacenters brings huge wastes of hardware resources and infrastructure investment. To improve resource utilization, a straightforward way is co-locating different workloads on the same hardware. To figure out the resource efficiency and understand the key characteristics of workloads in co-located cluster, we analyze an 8-day trace from Alibaba’s production trace. We reveal three key findings as follows. First, memory becomes the new bottleneck and limits the resource efficiency in Alibaba’s datacenter. Second, in order to protect latency-critical applications, batch-processing applications are treated as second-class citizens and restricted to utilize limited resources. Third, more than 90% of latency-critical applications are written in Java applications. Massive self-contained JVMs further complicate resource management and limit the resource efficiency in datacenters.

CCS CONCEPTS
- Computer systems organization → Cloud computing.

KEYWORDS
Resource efficiency, cloud computing, datacenter

1 INTRODUCTION

The huge cost of investment with low resource utilization has long been a great concern to cloud providers. Early in 2012, [24] revealed that the average CPU and memory utilization in Google’s production cluster were only 20% and 40% respectively. Nearly at the same time, the average CPU utilization of Amazon AWS EC2 was only 7% to 17% [17]. Since then, industry and academia have both devoted much effort to improve resource efficiency in datacenters [5, 7, 19].

However, the major obstacle to further improve resource efficiency is the performance interference brought by co-located workloads. The more workloads we co-locate on the same hardware, the more likely each workload suffers performance interference from each other [3, 19, 31]. The uncertainty of performance interference makes things even more complicated and highly reduces the Quality of Service (QoS) provided by cloud providers. How to improve the resource efficiency while guaranteeing the QoS of workloads becomes a great challenge to cloud providers.

To address this issue, some prior works [7, 9] attempt to exploit an efficient co-location and resource allocation strategy for applications before launching. One of the widely accepted strategies is prioritizing latency-critical (LC) applications and co-locating batch-processing applications to harness stranded CPU cycles [19]. Batch-processing applications could be deferred or evicted when any performance spike happens to the co-located LC applications. Some studies seek to adjust resource allocation dynamically during application’s execution, such as increasing the cache size of applications [19], or even rescheduling the tasks with lower priority when needed [7]. Resource allocation and dynamic adjustment methods are orthogonal techniques and can be adopted simultaneously in datacenters for higher resource efficiency.

In this paper, we ask the following question: after years of endeavor, how is it going in production datacenters now? How about the resource efficiency of today’s datacenters? Fortunately, Alibaba released a new Trace (AT) from their production cluster recently. The AT consists of 4K machines, 9K different services and 4M batch
jobs with static and runtime information over 8 days. Comparing to
the Google trace released in 2011, AT exhibits several new charac-
teristics and trends which we believe could shed some lights on the
resource management in datacenters, especially those confronting
Java workloads. Our key findings are summarized as follow:

**Memory becomes the new bottleneck of datacenters.** In
Alibaba’s cluster, memory seemingly becomes the new bottleneck
that prevents co-locating more workloads on one physical server,
whereas there still left considerable spare CPU cycles. The state-
of-the-art on improving resource efficiency of datacenters mainly
focus on CPU utilization of servers [6, 8]. However, we observe that
the average resource utilization of CPU and memory in AT turn out
to be 38.2% and 88.2% respectively, and some nodes nearly run out of
memory. As we further analyze the breakdown of memory utiliza-
tion in §4, we discover that Alibaba uses a conservative approach
to allocate resource to LC applications instead of fine-grained allo-
cation. LC applications reserve massive CPU and memory resource
in case of performance spike happens. Meanwhile, in each physical
host, memory is statically partitioned into several regions and appli-
cations (LC and batch) are only allowed to request memory in their
own regions. Due to the lack of efficient memory reclaim techniques,
the memory stranded by LC applications can not reallocate to other
jobs. The static memory management and conservative resource
allocation greatly affects the resource efficiency of datacenters.

**Batch-processing applications can only conservatively har-
est limited amount of resources.** Batch-processing applications
are always treated as second-class citizens in prior works [7, 19].
How about their de facto resource utilization in production data-
centers? We find out that they are subject to even worse treatments
in Alibaba trace. Batch-processing applications mainly follow three
principles to make compromise to LC applications. (1) They can
only utilize pre-defined limited amount of resource with lower pri-
ority. (2) They are the only applications suffering from rescheduling
in any case. (3) They are restricted to harvest limited amount of
resources in daytime, though there left remarkable spare resource.
The majority of batch workloads are launched at midnight, which
is of an order of magnitude larger than the amount of workload in
daytime. Those three principles directly put batch-processing appli-
cations in chains, which overprotects latency-critical applications
and hurts the resource efficiency of datacenters.

**Massive self-contained JVMs further complicate resource
management in datacenters.** In AT, more than 90% of LC and
some batch-processing applications are written in Java. Accord-
ingly, they will launch thousands of Java virtual machines (JVMs)
that are separated from each other and manage resource by their
own automatically. Co-locating so many self-contained JVMs will
cause at least two issues. (1) The unpredictable performance inter-
ference caused by the well-known garbage collection (GC) in JVMs
will exacerbate. Unfortunately, we can only overprovision enough
memory indirectly for each container to lower the frequency of
GC. It further reduces the resource efficiency. (2) The potentially
reserved but stranded memory resource will scale up, due to the
lack of efficient techniques to reclaim and reallocate memory in
JVMs to other applications. Since LC applications in Alibaba are en-
capsulated into containers, dynamically managing memory means
holistically coordinating between JVMs, containers and underlying
host systems to reclaim and reallocate stranded memory. It further
complicates resource management in datacenters.

In a nutshell, this work makes the following contributions:

- We analyze the characteristics of co-located workloads and
resource efficiency of Alibaba’s cluster from two perspectives,
resource allocation and adjustment.
- Based on the observation, we extract three key insights and
believe our findings will be applicable to future research.
- We introduces two metrics to quantify the compromise im-
pact of batch-processing applications, Reschedule Time Cost
(RTC) and Reschedule time cost Over execution Time (ROT).
- We propose a periodic GC adaptive mechanism to exploit
memory resource stranded by LC applications.

The rest of the paper is organized as follows: Section 2 introduces
background and related work. Section 3 presents an overview of
Alibaba’s new trace, cluster overall usage and workload character-
istics. Section 4 and 5 describe the resource efficiency of Alibaba’s
datacenter from two perspectives, resource allocation and adjust-
ment. Section 6 further discusses two topics about how to further
improve resource usage in datacenter.

2 BACKGROUND AND RELATED WORK

This section presents the motivation for workload characterization,
state-of-the-art cluster management systems and then describes
the architecture of Alibaba’s cluster management system.

2.1 Workload characterization

Co-locating multiple workloads on the same hardware is a straight-
forward approach to achieve high resource efficiency, but brings
severe resource contention [31]. Such contention leads to unpre-
dictable performance variability and highly reduces the QoS of
user-facing services [19]. Therefore, co-location makes things more
complicated in how to improve the resource efficiency while guar-
anteeing the QoS of workloads.

To better face this issue, it is necessary to understand the key
characteristics of workloads from real production environment. In
2011, Google released a public dataset from their production cluster,
presenting a 29-day trace of co-located workloads [25]. Reiss et
al. [24] thoroughly characterize the workloads in resource needs,
utilization and resource assignment. [1] investigates the resource
usage of Google cluster from user behavior. Besides, another line
of works characterize the workloads in VM-based datacenters. [5]
presents a detailed analysis of VM workloads behaviors from Azure.
[2] provides a view from private clouds in application characteris-
tics. These efforts provide a broad perspective and help us better
understand the various task shapes in large-scale datacenters.

In 2017, Alibaba, the largest cloud service provider in China,
released a publicly accessible dataset. It consists of 1.5K machines
that run both LC and batch-processing applications in a 12-hour
period [11]. Recent studies have analyzed the characteristics of
clustered datacenter from multiple perspectives. Lu et al. [20] analyzed
the imbalance phenomenon in the cluster. Liu et al. [18] focused on
the elasticity and plasticity of their semi-containerized cloud. And
[4] provided a unique view about how the co-located workloads
interact and interfere with each other.
In this paper, we analyse a new trace [12] released by Alibaba in late 2018 with larger scale and longer period. It’s the first publicly available dataset with precise information about the category of LC and batch applications. Based on the new trace, we provide a more comprehensive and microscopic view of the current practice about resource usage and resource allocation. Different from the prior works [4, 18, 20], this work mainly focuses on resource efficiency of Alibaba’s production cluster.

2.2 Cluster management systems

With the respect of availability and scalability, cluster management systems (CMS) play a key role in co-locating LC applications with batch jobs. A series of state-of-the-art CMSes like Borg [28] and Quasar [7] adopt an elaborate centralized scheduler for resource allocation and job placement. Two-level CMSes such as Mesos [10] uses a thin resource manager layer that allows multiple computing frameworks to share resources. Share state CMSes like Omega [26] allows multiple schedulers have a view of the entire cluster. Distributed CMSes such as Sparrow [23] makes random scheduling decisions for short batch jobs to achieve high throughput.

As for Alibaba, unlike the CMSes we discussed above, they use a hybrid of two-level and share-state architecture to manage resource and two different workloads. At the beginning, it’s online and batch applications are running separately in different clusters. The batch-workload cluster achieves high CPU utilization that more than 75%, while online-service cluster exhibits only about 10%, on average [13]. To improve resource efficiency, therefore, Alibaba co-located online services with batch jobs. As shown in figure 1, level1 has two schedulers: Sigma [13] and Fuxi [30], each one has its own resource pool. Sigma is responsible for managing user-facing, long-running online services, which are running in containers. Fuxi manages batch jobs, which are directly running on physical hosts. Meanwhile, in order to obtain global view and make better scheduling decisions, Sigma and Fuxi share the state of the entire cluster. This hybrid architecture has been verified to be a successful design by China’s biggest Shopping Festival in 2017 and 2018. It is worth noting that the co-located cluster is Alibaba’s internal private cloud rather than the public cloud Aliyun.

Because of some historical reasons, they did not redesign a new scheduler. Instead, they used a coordinated way to manage two independent schedulers and unified the resource pool. Level-0 controller is a central scheduler, which is responsible for resource management, data transfer and communication between Sigma and Fuxi. Level2 is on top of level1, consists of multiple business frameworks such as DB, search engine and compute.

3 CLUSTER TRACE OVERVIEW

This section describes the Alibaba’s new trace, overall cluster usage compared with the resource usage in 2017. And then presents the characteristics of workloads in co-located cluster that observed from the dataset.

3.1 Dataset

The new released trace [12] contains larger scale of machines and longer period. It consists of 4K machines, 9K online services and 4M batch jobs with static and runtime information over 8 days. The dataset contains three separate traces, i.e., servers, online services and batch jobs. Each trace contains two files, meta file records basic information and usage file stores the runtime data.

Servers. The dataset contains 4K machines. The meta file records machine ID, status, specifications and disaster level. Machine has two states, USING indicates that the machine is in using and IMPORT_INSTALLING means is being imported (e.g. installing operating system). Unlike the heterogeneity in Google’s machines [24], all server’s specifications are same in the trace with 96 cores and 1 unit memory normalized. The usage file records the runtime resource usage, including CPU, Memory, Network, IO and MPKI.

Online services. As an e-commerce company, Alibaba’s online services are mainly user-facing, long-running applications, such as search engine, online shopping and advertising. All of these services are running in containers and scaled across thousands of servers with strict Service Level Objectives (SLO) in tail latency. Each online service has two or more containers, which are scheduled on different hosts. Meanwhile, the containers of different services are co-located with each other on the same physical host. Besides, the majority of online services are written in Java, making up to more than 90% services and containers. It forms a huge Java applications cluster.

In the dataset, the meta file contains static information of containers such as status, its owner service, resource needs and the host where the container is running on. Container status includes Allocated, Started, Stopped and Unknown. Allocated indicates the container is in resource allocation stage; Started implies container is running; Stopped means it’s stopped; Unknown indicates some unknown errors happened. The usage file stores the runtime resource usage of containers, including CPU, memory, Network, etc.

Batch Jobs. Batch jobs such as MapReduce and machine learning are submitted by internal users. They are all non-production jobs, directly running on physical host. Batch jobs can be described by a ‘Job-Task-Instance’ model. Each submitted job will be split into multiple tasks with different computing logic. There are 12 different types of tasks and some of them can be expressed by a Directed Acyclic Graph (DAG) according to the computation dependency. Therefore, the completion time of a job is determined by all of
its tasks. Each task contains at least one instance, the smallest scheduling unit in Fuxi. Besides, each instance of the same task has same binary code and resource needs. Similarly, the duration of a task depends on all of its instances. This dataset includes 4 million batch jobs, 14 million tasks and 1.4 billion instances.

The meta file records basic information such as task type, start and finish time, DAG dependency, the amount of instance and planned resource. Each record in usage file indicates one instance execution information, including resource usage and final state. Failed state means the instance dose not run successfully; Interrupted implies it’s failed without completion; Ready indicates the it’s failed when ready to run; Running means it’s failed when running (such as terminated by user), while only Terminated state implies the instance is completed. An instance may fail due to various reasons (e.g. resource insufficient or machine failures). However, details are not provided in the dataset.

3.2 Overall Usage

We first analyse the overall resource usage of Alibaba cluster. From the dataset, all machines have the same specification of 96 cores and 1 unit memory normalized. Figure 2 shows the heatmap of machine-level resource usage over 8 days trace period. Each horizontal stripe corresponding to one machine, and memory seems achieve more higher resource utilization than CPU. From Figure 3 we can observe that the average CPU utilization of entire cluster is between 20%-50% in majority of the time, while average memory utilization is above 80% all the time and is almost fully utilized. Besides, we can find out the average CPU usage exhibits strong periodicity, which is clearly shown in Figure 4. Everyday it has a peak at 06:00 and then gradually declines, implying the accurate prediction is possible.

Different from the old trace in 2017, the new trace indicates Alibaba’s cluster achieved higher resources efficiency. Surprisingly the improvement of memory usage are much larger than CPU, from 50% [4] to more than 80%. At cluster level, the average utilization of CPU and memory are 38.2% and 88.2%, respectively. The cluster is nearly run out of memory. Besides, a common occurrence in the cluster is that a large number of batch jobs are queued with sufficient CPU resources, but they cannot be scheduled due to lack of memory1. It seems that memory shortage has became the main obstacle preventing co-locating more workloads on one physical server. And this phenomenon further limits the CPU efficiency of datacenter.

1This phenomenon is confirmed by Alibaba engineers.
3.3 Characteristics

**Workload heterogeneity.** The diversity applications are now a nature feature in modern datacenters [8] and it exacerbates the heterogeneity of workloads. We make the following observations: (1) There is a great variety of workloads of online and batch applications. In the dataset, batch jobs are submitted by internal users and the 'Job-Task-Instance' model highly increases the diversity of batch jobs. Moreover, there are 9K different online services, which are all end-user-facing applications. (2) The online services and batch tasks have various resource requirements, and the containers of the same service may configured with different specifications. The number of containers for each service is clearly shown in Figure 5(a). The thousands of applications make resource allocation an arduous task. (3) The various online services are with diverse SLO in tail latency. Based on the observations, the workload heterogeneity brings great challenge to resource allocation in co-located cluster, while guaranteeing the performance of online services and achieving high resource efficiency.

**Majority of batch jobs are small and short.** Data analytics frameworks such as MapReduce are running many short (seconds to minutes) jobs. In AT, majority of batch jobs are small and short with various resource needs. As shown in Figure 5(b), more than 50% of batch jobs, tasks and instances are finished within 10 seconds, while some are within sub-second. Therefore, Fuxi are designed to support high throughput and low scheduling latency, while ensuring placement quality. Besides, compared to the long-running online services, the high throughput of batch jobs brings high dynamic of cluster, which can further improve resource efficiency.

**Diurnal and nocturnal patterns.** Some studies have found out that user’s activity follows diurnal pattern and clusters received more requests and batch jobs at daytime [2]. In the dataset, online services and batch jobs also follow diurnal and nocturnal patterns. Because of user’s behavior, user-facing services are more active during the day, while batch jobs are reverse with high arrival rate at midnight. In order to improve resource usage, Fuxi will schedule more batch jobs to co-located cluster at midnight, when lower batch jobs during the day to protect LC application’s performance. As shown in Figure 5(c), batch job has a high arrival rate at 03:00 everyday and then gradually declines. Although diurnal and nocturnal patterns can better exploits the available resources in day level, it impacts the load varies of cluster. The uncertainty of user’s access and high throughput of batch jobs brings unpredictability in resource interference, which further complicates the resource management in co-located cluster.

4 RESOURCE ALLOCATION

As we analyzed in Section 3, memory seemingly becomes the new bottleneck and limits the resource efficiency of datacenter. In this section, we further decompose this phenomenon from the view of resource allocation.

According to the types of workloads running on, we partition the cluster into three regions, i.e., batch jobs only, LC applications only and co-located regions. To explore the impact of co-location on resource usage, we calculate the distribution of resource utilization for these three areas. As shown in Figure 6, we observe that batch only and LC only regions’ CPU utilizatons are significantly lower than that of co-located region. Batch only region’s average memory utilization is lower than others. Co-located area achieves higher resource efficiency. Because LC and batch jobs are all sensitive to resource allocations [3], how to allocate resource to them remains a problem. Conservative allocation will hurt the throughput for batch jobs, while optimistic allocation will cause unpredictable performance variability for LC services [19]. Below, we describe how Sigma and Fuxi allocate resource to LC applications and batch jobs, and static memory partition in detail. And then present how this hybrid allocation method limits the resource efficiency of datacenter.
4.1 CMS resource allocation

**Conservative allocation.** In order to guarantee the QoS of LC applications, a common approach is reserving ample resources for them [7]. Sigma takes this method to allocate sufficient resources for each container. It is a kind of over-provisioning and we called conservative allocation. As shown in Figure 7, we calculate the per-machine’s resource requested by online services. Resource of CPU has two values, one is request (requested by container) and another is limit (maximum CPU container can use). We observe that the sum of CPU cores requested by online services is almost equal to total capacity of cluster (94.2% average) and 37% memory resources are reserved by them. However, these applications’ CPU usage is very low (9.5% average) with only few moment achieve high efficiency. Containers’ memory utilization is higher (80.6% average), because of the feature of JVM. With the respect of resource efficiency, some servers’ CPU are oversold to improve usage. As a compressible resource, CPU can be effectively used by other jobs through CPU share or priority-based scheduling. Things are different to memory resource. In Alibaba, the majority of containers’ JVM configuration of minimum memory size for pile and heap (xms) and maximum memory size for pile and heap (xmx) are equal. Besides, due to the lack of memory reclaim technology, these memory will be always stranded in container’s lifecycle. Although when unpredictable bursty spike occurs, reserving sufficient resources can guarantee online service’s QoS, it hurts resource efficiency of cluster.

**Optimistic allocation.** Batch jobs can exploit the idle resources on host to improve usage and can use more than they request. They will request resources before scheduling and Fuxi uses an incremental resource allocation approach to meet their request [30]. However, as shown in Figure 8(c), majority batch instances actually use more resources in runtime than they requested, which brings high elasticity in cluster. Figure 8(a,b) depicts instances’ average runtime resource used vs. amount of resource they requested. We can find out that instances with low resources request tend to use more resources and most of highest average utilizations in each group are exceed 100% (used more than requested). This phenomena relevant to Fuxi’s incremental resource management protocol. It’s a kind of over-commitment and we called optimistic allocation. In this way, batch jobs can use the resources underutilized when online services are not busy, which is determined by Fuxi. Meanwhile, resources can be reclaimed by their owners when needed.

**Resource mismatch.** As discussed above, Alibaba’s CMS uses a conservative allocation method to protect online service’s QoS, while using a optimistic allocation approach for batch jobs to exploit the spare resources. However, behind these allocation, there exist a resource mismatch of CPU and memory, and memory first tends to be used up before CPU.

As for diverse online services, they only use a few types of static resource patterns. The resource request for online service includes CPU (request and limit) and Memory. The number of CPU request range from 1 to 32 cores (with 8 values in total), the CPU limit range from 1 to 192 (with 19 values) and the memory range from 0.0 to 25 (with 24 values in total, 25 means 25% memory resource of one physical host). Theoretically, there should be 3648 different resources combinations. However, only 61 patterns of (CPU Request, CPU Limit, Memory) are used observed from 9K services and 70K containers. Besides, (400, 400, 1.56) (400 means 4 CPU core) is most widely used resource pattern, covering 66.8% containers. With diverse online services, each application at most has two different patterns, meanwhile with low utilization. It indicates deprovision excess resource from LC applications and reallocate them to batch jobs is possible. As for batch instances, majority are CPU intensive and they have more flexibility to use spare CPU cycles and idle memory resources in host. However, batch jobs can not use the memory resources stranded by LC applications. In sum, given this hybrid allocation method, memory reaches the roof first. After years of endeavor, memory seemingly becomes the new bottleneck that prevents co-locating more workloads on one server.

4.2 Static memory partition

In Alibaba’s co-located hosts, memory resources are statically partitioned into three regions for LC applications, batch jobs and system. Figure 9 shows the memory model of co-located server. The solid line brackets represent the memory quotas of LC applications, batch jobs and system, the sum represents the total memory of the one server. In general, each machine has different quota ratio. However, LC applications generally would not request all memory in online
Who Limits the Resource Efficiency of My Datacenter

Figure 9: Static memory allocate model in co-located server. Quota ratios are various for different host.

Figure 10: Distribution of (a) memory occupied by latency-critical containers per-machine (b) and average memory utilization per-container.

Figure 11: The number of instance arrived by time and the number of instance reschedule happened in 15 minutes window over 8 days.

Quota and remains some idle memory. A part of them will be reserved as a buffer in order to prevent the bursty resource needs, and the remaining part can be used by batch jobs. But memory shortage is still a problem in some nodes, thereby infeasible for co-locating more batch workloads to further use idle CPU cycles. Figure 10 represents the distribution of memory requested by LC containers per-machine. 30%-60% of memory resource in majority servers are stranded by online services, while on average 30% of memory are underutilized. Furthermore, as discussed above, majority memory resource will be always stranded by Java applications and can not be reallocated to other jobs. Although static memory partition is temporarily a reasonable approach to assign resource, it wastes massive memory resource.

4.3 Implication

Sigma and Fuxi’s hybrid resource allocation and static memory partition methods bring high resource efficiency of cluster. However, it causes serious resource mismatch of CPU and memory and it’s not sustainable in the long run. Based on these observations, we argue the following. (1) Although conservative allocation can protect LC application’s performance, identifying the specific resources requirements of workloads is necessary to further improve resource efficiency. (2) Hardware resource disaggregation such as CPU and memory will bring great benefits. In Alibaba, they have deployed the compute and storage disaggregated infrastructure and achieve high efficiency. Besides, it brings high flexibility because resource allocation is independent from other components [27] and batch jobs can be reschedule to any machine without caring about data migration. (3) Better global perspective and elaborate scheduling algorithms are needed to improve cluster usage in Alibaba’s CMS. For example, there remains some idle machines in the batch only region when the majority of machines are busy.

5 RESOURCE ADJUSTMENT

As we discussed above, Sigma and Fuxi use conservative and optimistic allocation approaches for two types of workloads. However, due to the dynamic of cluster, the uncertain contentions on shared resources can significantly affect LC application’s performance and highly reduce the QoS provided by cloud providers. Therefore, some dynamic mechanisms are needed, such as feedback controller and dynamic adjustment in co-located cluster. From AT, we find both effective dynamic adjustment (+) and some shortcomings (-).

5.1 Batch job eviction and rescheduling

As second-class citizens, batch jobs are always make compromise to protect LC applications. Due to the unpredictable interference on shared resources, batch jobs could be deferred or evicted when any performance spike happens. In Alibaba’s cluster, Fuxi’s agent will halt batch instances first when online service’s performance is affected, and then reschedule the instances to another host if interference still exists. The uncompleted instances will retry multiple times until being completed successfully. For each retry, instance will be rescheduled to another host, which has sufficient resources, without data migration.\(^2\) In the dataset, batch jobs are only one suffering from rescheduling in any case. According to AT, although the rescheduling ratio (number of rescheduled instances over all instances in 8 days) is only 0.081%, the number of rescheduled instances (RIS) reaches 1 million and 20% of the RISs have rescheduled more than once. Furthermore, 61% of RIS are rescheduled after running a while, 38% are rescheduled without running and 3% belong to DAG tasks. Figure 11 shows the number of RIS and instance arrived by time over 8 days. We can observe that rescheduling occurs more frequently during the day when online services are more active, while infrequently at midnight when batch jobs have high arrive rate.

When instances are rescheduled to a new machine, they need to run from scratch. To quantify the impact of rescheduling on instance’s execution, here we define Reschedule Time Cost (RTC) and Reschedule time cost Over execution Time (ROT) as follows:

\[ RTC = ST_{LST} - ST_{FST} \]  

\[ ROT = \frac{RTC}{T_{exec}} \]

\(^2\)Because of the compute and storage disaggregated infrastructure in Alibaba, data is stored in other cluster.
with the feature of all-or-nothing, if the instance be halted and way, it can leverage the underutilized resource at night. Figure 14 when the online services are inactive, Fuxi will schedule more batch jobs to the co-location cluster to improve resource usage.

Taking a specific instance (ins_63835109 in the dataset) as an example, it was firstly scheduled on server m_1718 at timestamp 373421 and then ran 609 seconds without completion. After that it was rescheduled to server m_177 and began running at timestamp 374031 with execution time of 1196 seconds. We call 610 (374031-373421) seconds as RTC and 0.51 (610/1196) as ROT. This instance was delayed by 10 minutes due to rescheduling.

Figure 13 represents the CDF of ROT about all rescheduled instances, we observe that about 69.2% of RIS’s RTC are larger than 373421 seconds as RTC and 0.51 (610/1196) as ROT. This instance was delayed by 10 minutes due to rescheduling.

\[ \text{ROT} = \frac{\text{RTC}}{\text{ET}} \] (2)

where \( ST_i^{\text{LST}} \) refers the start time of instance i in final completed running, \( ST_i^{\text{FST}} \) represents the start time in first schedule running and \( T_{ET} \) indicates the execution time of the instance, as shown in Figure 12. RTC means the overhead of one instance because of rescheduling, the smaller the better. ROT indicates the percent of RTC in instance’s execution time, the smaller the better.

5.2 Nocturnal pattern+

In order to guarantee the QoS of online services, Fuxi and level0 strictly limit the resource utilization of batch jobs. Recall that Figure 5(c) shows batch jobs have obvious nocturnal pattern. At midnight, when the online services are inactive, Fuxi will schedule more batch jobs to the co-location cluster to improve resource usage\(^3\). In this way, it can leverage the underutilized resource at night. Figure 14 depicts the fluctuations of CPU, memory and disk I/O utilizations within 8 days. We observe obvious periodicity of diurnal variation. Every day, batch jobs have a high arrival rate at 03:00, and then CPU utilization increases dramatically and reaches the peak at 06:00, since there are many resource-consuming tasks running at that time. The nocturnal pattern greatly improve the resource efficiency at midnight. However, Fuxi schedules less batch jobs to the co-location cluster during the day to protect LC services, whereas there are ample resource.

5.3 Memory stranding-

Alibaba has a large number of online services to provide real-time services. In the dataset, more than 90% of online services are Java applications. Due to the characteristics of Java Virtual Machine (JVM) in memory usage\(^2\) and conservative allocation, online services reserve a large amount of resources at the beginning and strand them in their lifecycle\(^4\). Figure 10 shows the distribution of memory stranded by containers per-machine and memory utilization. In each server, the memory allocated to the containers is typically between 40% and 60%, and most containers’ memory utilization are more than 90%, consuming over 40% of cluster memory resources.

By analyzing the memory stranded by LC applications we find that most of memory are consumed by JVM heap. JVM heap memory consists of two parts: Young Generation and Old Generation. Young Generation is the place where all the new objects are created. When young generation is filled, Minor garbage collection (Minor GC) will be performed. The frequency of Minor GC reflects the real memory requirements of JVM as well as the online workloads. Figure 15 reveals container’s Minor GC interval. The online containers have a longer interval of Minor GC from midnight to early morning, which means they require very little memory at that time. When the memory demand of JVM is declined, the JVM heap memory size remains constant, causing plenty of memory are stranded. But during this period, batch jobs need more memory to support the execution of millions of tasks. Therefore, self-contained JVM and lack of reclaim technique further limit the resource efficiency of datacenter.

5.4 Implication

Dynamic adjustment and nocturnal pattern are both effective approaches to better leverage idle resources in cluster. Although Alibaba has made great endeavor, it still has some deficiencies. Based on these observations, we argue the following. (1) Memory reclaim

\(^3\)In Alibaba, besides co-location cluster, it still has batch-only and online-only cluster. Only part of batch jobs can be scheduled to co-location cluster and others are in batch-only cluster.

\(^4\)Sigma not fully support real-time resource adjustment, container migration and reschedule.
Who Limits the Resource Efficiency of My Datacenter

![Graph showing Minor GC intervals](image)

**Figure 15:** Overall Minor garbage collection interval within 8 days. Lower interval indicates containers consume memory more frequently.

...technique is needed to improve resource usage. A large amount of memory can be reclaimed from online services at midnight and reallocated to batch-processing jobs. (2) Targeted optimization is necessary for such a large-scale JVM cluster, such as dynamic management of JVM heap memory. (3) Capacity planning can be better optimized according to the diurnal pattern of online services and prediction information.

6 DISCUSSION

Aiming at achieving high resource efficiency of cluster, Alibaba co-locates batch-processing jobs with online services to leverage idle resource. However, hybrid resource allocation and adjustment further complicate resource management in datacenters. Recent research works reveal that the ratio of computing-to-memory-capacity is becoming more imbalanced [14, 15]. The utilization of CPU and memory from AT also confirms this observation (Figure 14). [16] has pointed out that the increasing imbalance between computation and memory may cause memory capacity per-core dropping by 30% every two years. Therefore, the performance of future systems will be limited because of inadequate memory capacity and new memory wall will emerge in co-located cluster. Below, we will discuss from two perspectives, JVM optimization and hardware resource disaggregation, to address this imbalance.

6.1 Periodic GC adaptive adjustment

As discussed in section 5.3, a large amount of memory resources are straddled by LC applications due to the conservative allocation and the feature of JVM. Since the performance of garbage-collected applications is highly sensitive to heap size [29], applications with larger heap size will have low frequency of garbage collections (GC). However, it limits resource efficiency in datacenter and will cause thrashing [21]. Based on this observation, we propose a mechanism of Periodic GC adaptive adjustment in JVM. It is an effective solution to maintain an appropriate heap size and it will set a fixed GC frequency for JVM. If a Java application is inactive with few GC, it will actively perform GC operation to reclaim stranded memory, and we will be able to reallocate such memory resource to batch jobs at specific times (e.g. 03:00-06:00). This mechanism can increase the throughput of batch tasks and overall resource usage of cluster.

To quantify the benefit of this mechanism, firstly we design a formula to represent the relationship among workload’s CPU utilization, Young Generation Size (YGS) and Minor GC Interval (MGCI) as follows:

\[ CPU = k \cdot \frac{MGCI}{YGS} + b \]  \hspace{1cm} (3)

Where CPU indicates the CPU utilization, MinorGCInterval and YoungGenerationSize are values corresponding to CPU, and k is a constant. According to the formula, the fine-grained control can provide 3.43% of memory for each physical machine.

Several assumptions need to be made before going into detail:

1. Under the same workload (CPU utilization), the interval of JVM’s Minor GC is linearly related to Young generation size.

\[ YGS = k \cdot \frac{MGCI}{CPU} + b \]  \hspace{1cm} (2)

2. Theoretically, when the size of the young generation size approaches zero, the interval of Minor GC will also approach zero, thus b = 0.

\[ CPU = k \cdot \frac{MGCI}{YGS} \]  \hspace{1cm} (3)

15 seconds is an empirical value for periodic GC adaptive adjustment.

![Equation showing formula for CPU utilization](image)

According to the online services’ JVM trace, the memory size of young generation accounts for 5.49% of one physical host memory and the minimum GC interval (Min) is larger than 40 seconds from 03:00 to 06:00. Assuming that the Minor GC interval is compressed from 40s to 15s, the young generation size can be compressed to 2.06% according to the formula above. This indicates that 3.43% memory resource in one host can be reallocate to other jobs. Taking a cluster with 4,000 machines (256G memory each) as an example, about 34.3T memory can be provided in the certain time, which is equivalent to a capacity of 137 servers.

6.2 Hardware resource disaggregation

As discussed in section 4, hybrid resource allocation method causes severe resource mismatch of CPU and memory. Machine, as the natural boundary of hardware resource, has become the bottleneck to achieve higher resource utilization in datacenters. For example, a container can only use CPU and memory resource in the same physical host. With the increasing heterogeneity of applications, the traditional monolithic server [27] is reaching its limits. Separating the monolithic server into several independent hardware components like computation, storage and networking will greatly improve resource utilization and elasticity. Each component has its own resource controller and resource packing will become more efficient, since resource will not be restricted to the same machine. Thanks to the advances network technologies, it becomes common to break the storage components from traditional computer architecture with more cost-effectively and it has been used in Alibaba’s co-located cluster. Batch instances can be scheduled on any servers with sufficient resources without caring about data migration.

Since the resource mismatch of CPU and memory heavily limits the resource efficiency in Alibaba’s cluster, the hardware resource disaggregation in CPU and memory will bring following the benefits...
to their cluster. (1) Due to the disaggregation, memory and CPU resource requests can go beyond the boundary of a physical host. In fact, they can be satisfied by the entire cluster, and can make resource packing more efficiently. (2) Memory fragment will be rarely seen and resource usage will become more effective.

7 CONCLUSION AND FUTURE WORK

This work has made a thorough analysis of newly released dataset from Alibaba’s production cluster. To better understand the resource efficiency and characteristics of workloads in their datacenter, we breakdown into two perspectives, resource allocation and resource adjustment. The result shows several findings as follows. a) Resource mismatch between CPU and memory is acute in their cluster, and memory becomes the new bottleneck. b) Batch-processing jobs are treated as second-class citizens and always make compromise to protect latency-critical applications’ performance. c) Hybrid of conservative and optimistic allocation method and massive self-contained Java application further complicate resource management. Overall, we believe our findings will be applicable to system designers and provide broad visibility for future research.

8 ACKNOWLEDGEMENT

We would like to thank anonymous reviewers for their valuable feedbacks and suggestions. We thank Alibaba’s engineers Guoyao Xu, Cheng Wang, Jie Chen, Shun Lv, Xiaoyu Zhang, Yinghao Yu for their feedback on the paper.

This work was supported in part by National Key R&D Program of China (2016YFB1000201), and the National Natural Science Foundation of China (Grant No. 61420106013 and 61702480), and Youth Innovation Promotion Association of Chinese Academy of Sciences (2013073) and Alibaba Group through Alibaba Innovative Research (AIR) Program.

REFERENCES


