Exploring Imbalances among Microservice Containers in Large Cloud Platforms

Chenyu Lin
University of Macau
mc14889@um.edu.mo

Shutian Luo
Yale University
shutian.luo@yale.edu

Huanle Xu
University of Macau
huanlexu@um.edu.mo

***Abstract*—As software systems evolve, monolithic applications are often transformed into a collection of lightweight and loosely-coupled microservices. Understanding the intricacies of microservices is crucial for optimizing their deployment. Unfortunately, existing research and literature often fail to provide a comprehensive analysis of microservice performance and resource utilization, particularly when it comes to addressing performance imbalances across microservice containers within large-scale production clusters.**

This paper aims to address a critical research gap by conducting an extensive analysis of performance imbalances within Alibaba's clusters, utilizing response time as a key indicator. Additionally, we investigate the issue of utilization imbalance. Our study reveals that a substantial number of microservices suffer from performance imbalances, primarily attributed to disparities in node resource utilization and uneven communication overhead resulting from inter-node requests. Furthermore, we identify that offline jobs significantly contribute to the imbalanced utilization of node resources. Based on our findings, we propose a novel joint optimization designed to minimize inter-node requests. By adopting this optimization, we aim to enhance the end-to-end performance of the system.

1. Introduction

Nowadays, microservice architecture is becoming increasingly prevalent [5], [9], [12], [13], [20], [21], [22], [23], [25], [33], [34], due to its ease of maintenance, cross-team development, friendly deployment and so on. Currently, leading cloud service providers like AWS and Alibaba have begun to offer an off-the-shelf microservices architecture for users, simplifying their application deployment processes [1], [4], [6].

In order to effectively utilize microservice architecture, several benchmarks have been developed to explore its key characteristics, including Train Ticket [35], Acme Air [30], μSuite [27], and DeathStarBench [11]. These benchmarks offer comparisons between microservices and monolithic applications, focusing on network overhead [30], the efficiency of remote procedure calls [27] and the impact on performance implications on and hardware performance [11]. Additionally, a microservice simulator has been developed to clone micorservice based on their runtime metrics [14], thereby replaying the characteristics of microservice. These studies only provide insights into relatively small-scale clusters, and their results do not necessarily apply to production environments. The existing trace analysis on microservice [18], [19] mainly focus on analyzing graph dependencies between microservices, while comprehensive analysis of microservice imbalances remains limited.

In production clusters, it is common for a microservice to be deployed across hundreds of containers [18], [19]. From the end users' standpoint, performance imbalances among containers of the same microservice can significantly contribute to high variances in response time (RT), ultimately leading to long tails and violating Service Level Agreements (SLAs). From the perspective of service providers, an imbalance in resource utilization among microservice containers necessitates the deployment of additional containers, thereby diminishing the overall efficiency of the cluster. Consequently, there is a crucial need to thoroughly investigate and address imbalances among microservice containers.

This paper presents a comprehensive analysis of microservices deployed in Alibaba clusters, with a specific focus on identifying imbalances in both performance and resource utilization. Our investigation encompasses a wide range of imbalances, including those at the container and physical machine (referred to as "node") levels, as well as workload imbalances. In the meanwhile, we pay particular attention to performance imbalances in terms of response time (RT) across containers of the same microservice. We highlight the significant impact of node utilization imbalances and inter-node communication on RT performance. Based on our analysis, we propose a novel joint optimization that aims to reduce inter-node communication. We illustrate our studies and the important findings in the following.

CPU utilization among physical servers in Alibaba clusters exhibits a significant and periodic imbalance, reaching up to 35%, primarily due to short-term offline jobs. Given that online services typically experience lower activity levels around midnight, a large number of offline jobs are scheduled into clusters to optimize resource efficiency [17]. These jobs, often short-term, can easily cause imbalance on resource usage among nodes (Section 3).

Load balancing across microservice containers can be attained even when an upstream microservice connects to only a limited subset of downstream microservices.

The Alibaba cluster employs a long-lived connection RPC between microservices to reduce the overhead of reconnections. To prevent an overwhelming number of connections between microservices, an upstream microservice is designed to connect only to a small set of downstream microservices. The characterization results demonstrate that load balancing can be effectively achieved on a global scale.

Performance imbalances in microservices is heavily related to uneven resource utilization across physical nodes. Though it is possible to maintain load balance among containers of each microservice, performance disparity among these microservice containers remains a concern. This variation is intensified in the end-to-end (E2E) latency, particularly as a single request can trigger a sequence of calls between multiple microservices. Consequently, to mitigate these performance imbalances across microservices, it is crucial to formulate a more refined strategy for the development of a sophisticated, unified resource orchestrator [17]. This orchestrator should ensure balanced resource utilization across nodes within the cluster.

Effective scheduling and load-balancing policies can lead to a remarkable improvement of up to 75% in microservice RT. By minimizing communication overhead, which can be substantial for lightweight microservices, careful co-locating scheduling and load-balancing policies can effectively enhance overall performance. Specifically, optimizing the co-location of microservice pairs that handle significant communication traffic can yield substantial performance gains. Thus, it is crucial to develop an intelligent strategy that harnesses the power of co-location to achieve high RT performance.

To summarize, we have made the following contributions in this paper:

- We present a comprehensive study focusing on resource utilization (Section 3), workload (Section 4), and microservice performance (Section 5) in large-scale production clusters.
- We provide an in-depth analysis of microservice co-location, yielding valuable insights into microservice scheduling and resource management strategies (Section 6).
- We propose a new joint optimization idea that combines scheduling policy and load-balancing policy to improve the end-to-end performance of the system (section 7).

2. Background

2.1. Microservice Architecture

Microservice architecture represents a loosely coupled architectural approach designed specifically for cloud-native applications. This approach involves breaking down a traditional monolithic application into multiple small, independently deployable microservices. This division allows for a more granular management of the application. It is worth noting that each microservice within the application

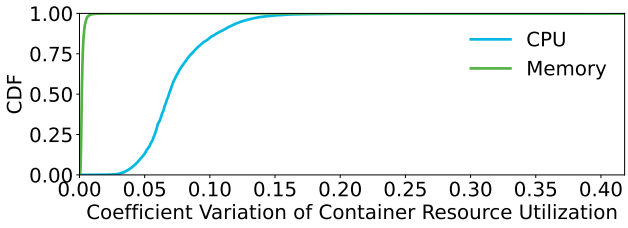


Figure 1: CDF of Coefficient Variation for container CPU (memory) utilization of a microservice.

can be executed in multiple containers, and each container is assigned to a physical machine referred to as a *node*, which may accommodate multiple containers. By dynamically adjusting the number or size of containers, the system administrator can easily allocate resources to a particular microservice without impacting other components.

Response time. The E2E latency of a service stands as a paramount metric for user satisfaction. Typically, service providers adopt the 99^{th} or 95^{th} percentile E2E latency of all requests as a SLA. The E2E latency of a request comprises the cumulative RT of the microservices involved in processing the request. Each microservice's RT is defined as the time elapsed from when it receives the request to when it returns the response. In order to minimize the overall E2E latency of a service, system managers must reduce the RT of each microservice by augmenting their allocated resources or other methods.

2.2. Alibaba Trace Overview

Alibaba offers a comprehensive dataset [3] spanning a duration of 13 days, obtained from two hybrid scheduling clusters proficient in handling both online and offline tasks. This dataset comprises a wide array of data derived from thousands of microservices and nodes, providing a representative and up-to-date depiction of a typical microservice cluster. It should be noted that certain data in the dataset has undergone normalization, rendering absolute values unattainable. Alibaba monitors microservice performance at the container level [2], capturing data at one-minute intervals. This data includes both Query Per Second (QPS) and RT metrics.

3. Resource Utilization Imbalance Analysis

As introduced in Section 1, service providers aim to enhance cluster resource utilization while guaranteeing that SLA requirements are met for all users. In Section 2.1, we present that a microservice may encompass hundreds of containers spread across diverse nodes. It is noteworthy that uneven resource utilization among containers can have a substantial impact on microservices [19]. This imbalance becomes particularly critical when considering SLA requirements and total resource efficiency. Specifically, imbalanced performance typically generates long-tail RT distribution for individual microservices and therefore make service

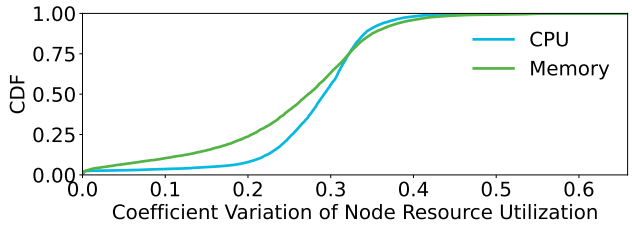


Figure 2: CDF of Coefficient Variation for node CPU (memory) utilization.

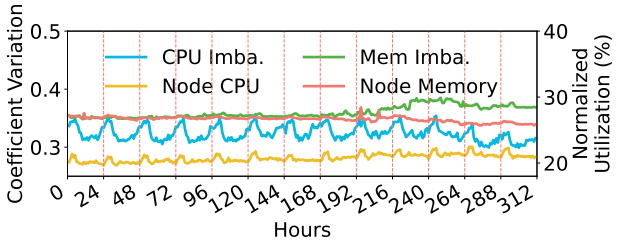


Figure 3: The imbalance of node utilization undergoes temporal variations.

E2E latency violates SLA requirement. Prompting service providers to allot supplementary resources, and thus reducing resource efficiency. Our analysis conducted on Alibaba's data revealed the following findings: (i) nearly 80% of microservices experience up to 1.25× resource utilization imbalance at the node level, and (ii) there is a conspicuous periodicity regarding CPU utilization imbalance at the node level.

3.1. Spatial Analysis

3.1.1. Resource utilization across containers. In order to assess the imbalances of resource utilization among containers of each microservice, we use the coefficient of variation c_v as a metric, which can be expressed as follows:

$$c_v = \sigma/\mu. \tag{1}$$

Here, σ and μ denote the standard deviation and mean of a given dataset, respectively. We calculate the c_v based on the resource utilization data of all containers belonging to the same microservice at a given moment. In Fig. 1, we present the cumulative distribution function (CDF) of the c_v for CPU and memory resource utilization. The CDF demonstrates that 80% of microservices have a c_v value of 0.06 or less for CPU utilization, indicating minimal variation. Furthermore, the majority of microservices exhibit negligible variation in memory utilization, suggesting well-balanced container resource utilization.

Alibaba maintaining a well-balanced resource utilization of containers by equally distribute workload among all containers of a microservice. Designing such a workload balancer for large-scale clusters is a complex task, in Section 4, a detailed discussion is provided regarding the interesting load-balancing mechanism utilized by Alibaba.

3.1.2. Resource utilization across nodes. There is a notable imbalance in resource utilization among nodes where the same microservice is deployed. As each container of a microservice is assigned to a specific node, different containers of the same microservice may experience varying levels of resource interference on their respective nodes. To evaluate this imbalance, we calculate the c_v based on the resource utilization data of all nodes deployed with the same microservice at a given moment. Fig. 2 illustrates that 80% of microservices may suffer from a c_v of more than 0.25 in node CPU utilization. Additionally, 80% of microservices experience a c_v of no less than 0.15 in node memory utilization, which is comparatively lower than that of the CPU utilization. This substantial variation can lead to unbalanced resource interference, causing a severe impact on container performance.

Scheduling strategy of Kubernetes is one of primary reasons behind this phenomenon, it schedules containers to nodes based on the capacity of nodes, without considering the actual utilization of the nodes. This lack of consideration for utilization contributes to the observed imbalance phenomenon. Furthermore, [19] presented that hybrid scheduling is widely adopted and cluster manager often co-locate batch jobs with online services on the same node. Batch job containers usually occupy much more resources, while containers from online services only consume a small amount of resources [17]. These differences in resource consumption patterns further exacerbate the overall imbalance in resource utilization among nodes.

This imbalance in resource utilization also presents opportunities for optimizing resource allocation. System managers can improve container performance by strategically placing them on low-utilization nodes.

3.2. Temporal Analysis

3.2.1. Node utilization over time. Another notable observation is the presence of a distinct periodicity in the variation of node CPU utilization. Fig. 3 illustrates the imbalance in resource utilization, as measured by the c_v, across different hours. Each vertical red line denotes midnight, representing a new day. The average CPU and memory utilization imbalances for each hour are depicted by the blue and green lines, respectively. Additionally, the normalized CPU and memory utilization are represented by the yellow and red lines, respectively. Analysis of the figure reveals that the imbalance of CPU utilization escalates between the hours of 10 p.m. and 6 a.m., coinciding with a corresponding increase in node CPU utilization during the same time period. In contrast, the memory utilization and its imbalance remain relatively stable throughout and do not exhibit any discernible periodic characteristics.

The observed CPU utilization imbalance can be attributed to the presence of batch jobs during the designated time period. This relationship is demonstrated in Fig. 4, which displays the workload pattern over time. It is evident that the workload experiences a significant drop from 10 p.m. to 6 a.m. Accordingly, Alibaba cluster manager taking

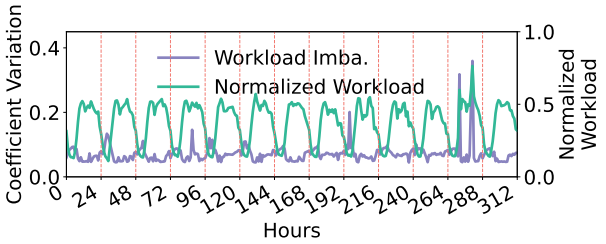


Figure 4: The imbalance of workload undergoes temporal variations.

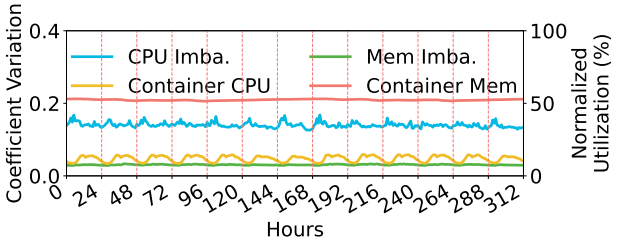


Figure 5: The imbalance of container utilization undergoes temporal variations.

advantage of this period of reduced workload, strategically schedules more batch jobs during these hours. Consequently, the higher CPU utilization and subsequent node utilization imbalance can be attributed to this deliberate utilization of computational resources during the specified time period.

3.2.2. Container utilization over time. Contrarily, the utilization of container resources exhibits a stable pattern over time. Fig. 5 illustrates the variation in container resource utilization as time progresses. In contrast to the node resources utilization, the container resource utilization does not display significant periodic characteristics.

4. Workload Imbalance Analysis

The balanced utilization of container resources of each microservice, previously discussed in Section 3, implies a fundamental equilibrium among the workloads assigned to all containers. In this section, we conduct a comprehensive analysis of the workload distributed across all microservice containers to understand Alibaba's load balancing mechanism. Our investigations suggest that UM containers strategically initiate connections with around 5% of DM containers only in Alibaba's cluster. Despite this selective connectivity approach, the workload is efficiently distributed among the DM containers, thus assuring a balanced workload distribution across the containers.

4.1. Spatial Analysis

In a microservice system, UM will send requests to DM containers for processing, these requests are considered as the workload for DM containers. Load balancers will determining how to distribute the workload across different containers. Alibaba provides QPS data for individual containers, allowing us to analyze the workload of containers

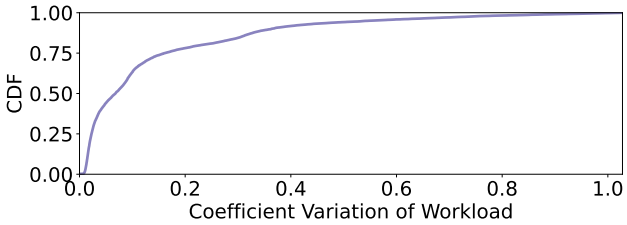


Figure 6: CDF of Coefficient Variation for container workload of a microservcice.

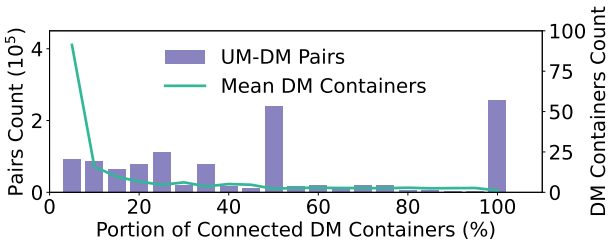


Figure 7: Distribution of UM-DM pairs based on different percentage of connected DM containers.

of the same microservice and assess the degree of workload imbalance present in that microservice. We employ the c_v mentioned in (1) and gather QPS values of all containers belonging to the same microservice at a given moment as the dataset.

In Fig. 6, we present the CDF of the c_v of workload across containers of each microservice. The results indicate that approximately 80% of microservices exhibit a c_v less than 0.2 for workload, which suggesting the workload of most microservices is predominantly evenly distributed in DM containers. However, maintaining an even distribution of workload in a large-scalse cluster is a significant challenge. Currently, RPC is one of the most widely used protocols for communication between containers in a microservice system. It is built on top of the HTTP/2.0 protocol [8], which inherently supports long-lived connections. Consequently, in a microservice system, UM containers will often establish long-lived connections with DM containers.

In a large-scale microservice system, a DM may be used by hundreds of UMs, each of which may own hundreds of containers. However, due to resource limitations, DM containers can't maintain connections to all UM containers, and this can lead to service unavailability. To figure out how Alibaba addresses this challenge, we count and analyze the connections between UM and DM containers by tracing their requests. If a UM container does not send any requests to a particular DM container, it is assumed that there is no connection between them. In Fig. 7, we illustrate the distribution of the average percentage of connected DM containers for every UM-DM pair. The graph reveals that a significant number of UM-DM pairs utilize half or all of the available DM containers. Conversely, most of other UM-DM pairs prefer to use within 40% of the DM containers. Additionally, the cyan line represents the mean number of DM containers for different utilization of DM containers. It is evident that as the number of DM containers increases,

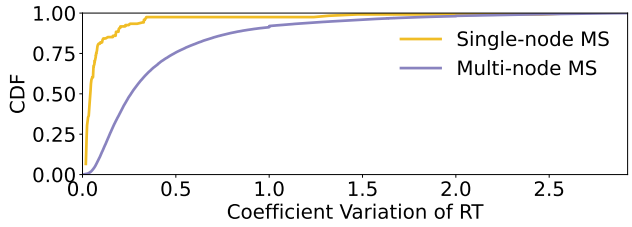


Figure 8: CDF of Coefficient Variation for container performance of a microservcice.

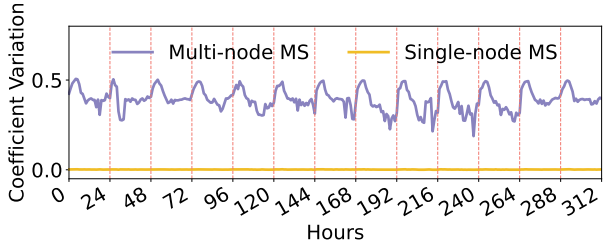


Figure 9: The imbalance of performance undergoes temporal variations.

the percentage of utilized DM containers decreases. Notably, when there are approximately 100 DM containers, UM containers tend to connect to only 5% of them.

Based on the previous analysis, we can now examine Alibaba's workload balancing technique. In this technique, each UM container establishes connections with a subset of DM containers, and evenly distributing requests among them. Furthermore, a load balancer is employed to manage the connections of each UM container, ensuring an overall workload balance across the DM containers. This design effectively addresses the issues of workload imbalance and excessive connections. However, it necessitates meticulous design considerations for the load balancer and introduces heightened complexity to the network topology.

4.2. Temporal Analysis

It is worth noting that the workload itself exhibits strong periodicity, whereas its imbalance does not. As depicted in Fig. 4, the cyan line represents the normalized workload, while the purple line represents the imbalance of workload. During the time period from 10 p.m. to 6 a.m., the workload diminishes to approximately one-third of its peak value, while the workload imbalance remains consistent.

At 272 and 281 hours, we observe the presence of two peak workloads, which are 33% and 40% higher than the normal workload, respectively. During these two hours, the workload imbalance also increases. One may assume that the imbalance is caused by new containers that deployed to handle the peak workload, which may need time to warm up and establish connections. However, we have examined the number of containers during these hours and found that there are no changes. Since the workload imbalance is not attributable to the addition of new containers, it is likely that the load balancer is responsible for this imbalance, as it appears to struggle in managing extremely high workloads effectively.

5. Performance Imbalance Analysis

As discussed in Section 1, RT imbalance can easily lead to SLA violations, causing dissatisfaction among end-users. Therefore, we delve into a container-level analysis of performance imbalance in this section, focusing on the RT of individual containers. Our findings reveal a considerable performance imbalance among containers belonging to the same microservice as much as 1.25×. Moreover, this performance imbalance becomes even more obvious when the containers are assigned across multiple nodes, as much as 1.5×. Additionally, we have observed that the performance imbalance exhibits periodicity and is closely associated with imbalances of node utilization.

5.1. Spatial Analysis

In the Alibaba trace data, mean RT values are provided for individual containers in one-minute intervals. Fig. 8 illustrates the CDF of performance imbalance for microservices, measured by the c_v of RT. The yellow line represents microservices that have all containers located on a single node, whereas the purple line represents microservices with containers assigned across multiple nodes.

Building upon the findings from the resource analysis (Section 3) and workload analysis (Section 4) conducted in Alibaba clusters, it is evident that resource utilization and workload balancing between containers are effectively managed. However, we have identified another critical issue that significantly impacts performance imbalances: the placement of microservice containers. Fig. 8 visually represents this phenomenon. When microservices are distributed across multiple nodes, approximately 25% of them exhibit a c_v for response time (RT) greater than 0.5, indicating a notable imbalance. In contrast, when all containers of a microservice are located on a single node, as much as 90% of them demonstrate a c_v value within 0.25 for RT, highlighting a significantly lower level of imbalance.

This RT imbalance presents opportunities for further optimization of resource allocation and scheduling algorithms. In current microservice systems, there is a focus on the 99^{th} or 95^{th} percentile of RT in general. By balancing the performance between containers, it is possible to reduce the 99^{th} and 95^{th} percentile of RT, even if the overall resource usage remain unchanged.

5.2. Temporal Analysis

We have also conducted an analysis of performance imbalance from a temporal perspective, as depicted in Fig. 9. The yellow and purple lines represent the c_v, illustrating the performance imbalance of single-node microservices and multi-node microservices, respectively. The graph reveals that the performance imbalance of multi-node microservices exhibits strong periodicity. It tends to increase from 10 p.m., reaching its peak at 2 a.m.

When comparing this result with node resouce imbalances illustrated in Fig. 3, a correlation emerges between

the imbalances in performance of multi-node microservices and node CPU utilization, indicating that performance imbalance are influenced by imbalance in node CPU utilization. Conversely, the performance imbalance of single-node microservices exhibit stability and remain unperturbed by imbalance in node CPU utilization. In the following section, we offer a potential solution to reduce performance imbalance by strategically selecting suitable nodes for container assignment.

6. Characterizing the Key Factors behind Performance Imbalances

In Section 5, we have demonstrated the considerable performance imbalance is mainly caused by the location of microservcice containers. This location has a heavy impact on two factors: (i) variations in resource utilization among nodes, and (ii) network latency. These two factors can greatly affect performance and thus lead to performance imbalances. In this section, we will delve into how and to what extent these two factors affect performance.

6.1. Unbalanced Resource Interference

In Section 3, we have demonstrated that containers belonging to the same microservice may experience varying levels of resource interference on different nodes. To analyze the correlation between node resource utilization and container performance, we employ the Pearson Correlation Coefficient (PCC), which is defined in the following formula:

$$\rho_{rp} = \frac{\sum_i^n (r_i - \bar{r})(p_i - \bar{p})}{\sigma_r \sigma_p (n-1)}, \tag{2}$$

where r and p represent the resource utilization dataset and the performance dataset of a microservice, respectively. Both datasets have a size of n. The mean and standard deviation of r and p are denoted as $\bar{r}$, $\bar{p}$, σ_r, and σ_p, respectively. The value ρ_{rp} represents the correlation between resource utilization and performance.

We still utilized RT as the performance indicator. Fig. 10 illustrates the CDF of the correlation between performance and CPU/memory utilization in blue and green lines respectively. The correlation between node CPU utilization and container performance is found to be highly significant. Approximately 50% of microservices exhibit a correlation coefficient larger than 0.75 between performance and node CPU utilization. On the other hand, the correlation between node memory utilization and performance is relatively low, but still considerable. There are around 20% of microservices demonstrate a correlation coefficient greater than 0.5 in this regard.

Given the strong relationship between resource utilization and performance, it is crucial for system managers to prioritize node interference when assigning containers to nodes. The current Kubernetes scheduler focuses on balancing the overall allocated resources across nodes but does not consider the actual node resource utilization. If cluster managers can achieve a balance in resource utilization among nodes, containers are more likely to exhibit balanced performance.

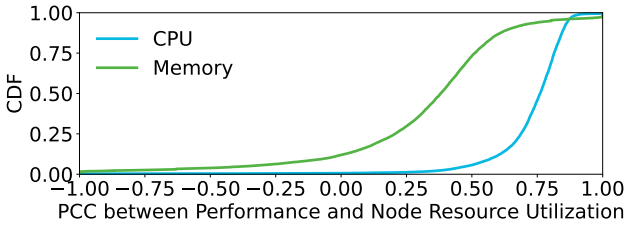


Figure 10: CDF of PCC between resource utilization and performance.

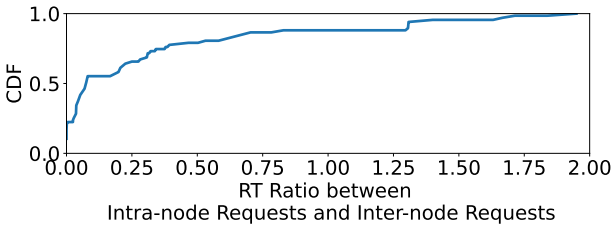


Figure 11: CDF of the RT ratio between intra-node requests and inter-node requests.

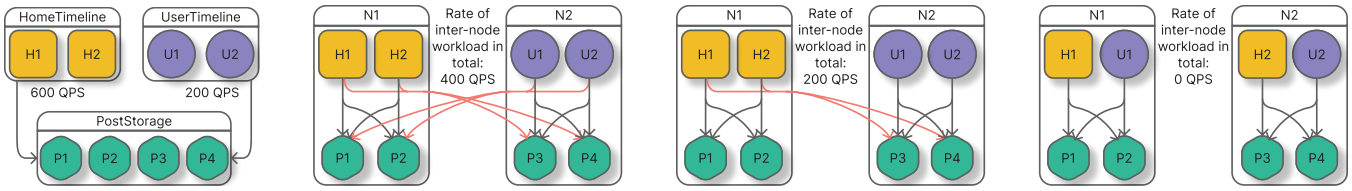
Furthermore, as illustrated in Fig. 10, it is evident that different microservices display varying levels of resource sensitivity. Notably, around 30% of microservices demonstrate a lesser degree of sensitivity towards memory. Consequently, assigning these containers to nodes with high memory utilization would not degrade their performance too much. This strategic assignment approach can reduce service E2E RT and optimize the overall performance of containers.

6.2. Network Latency

When two containers are assigned to the same node, the network latency associated with data exchange between them is minimal. However, when one container needs to communicate with another assigned to a different node, the inter-node network latency can considerably impact performance. To quantify this cost, we examine the RT of DM for both inter-node and intra-node requests made by the same UM-DM pairs. It is crucial to underscore that while calculating the RT of DM, it is necessary to exclude the RT contribution by the "DM of DM" to ensure accurate measurement. Such a quantification can be achieved by analyzing the requests' *rpcID* and *traceID*.

In Fig. 11, we present the CDF of ratios between intra-node RT and inter-node RT of the same UM-DM pairs. The graph illustrates that approximately 80% of requests experience a significant RT decrease of more than 50% when they are intra-node requests. However, even though some requests are intra-node requests, they exhibit higher RT.

There exist two optimizations regarding the issue of network latency: (i) Packing UM containers and their frequently accessed DM containers together and placing them on the same node. (ii) Directing UM workload to intra-node containers as much as possible. In addition to these optimizations, it is crucial to address resource utilization



(a) (b) (c) (d)

Figure 12: (a) Two UM microservices send requests to a single DM microservice with different rates. (b) Fully Randomized. Inter-node workload is 400 QPS. (c) Random Assignment, Workload Aware. Inter-node workload reduced to 200 QPS. (d) Assignment Aware, Workload Aware. Absolutely no inter-node workload.

imbalances among containers and nodes, which presents a significant challenge. To tackle this challenge, we have proposed a new optimization idea, which will be presented in the following section.

7. Joint Optimization for Minimizing Inter-node Communication

As discussed in Section 6, the deleterious impact of inter-node network latency on container performance is conspicuously evident. This highlights the need for a carefully crafted optimization aimed at minimizing inter-node communication. However, this endeavor is inherently complex and requires the harmonious coalescence of multiple policy paradigms, including load balancing and scheduling policies. The confluence of these multiple facets within a comprehensive optimization model presents formidable challenges. Consequently, this section offers a basic conceptual framework that emphasizes the central importance of such a joint optimization. Subsequently, experiments are conducted to substantiate the efficacy of various policies.

7.1. Experiment Setup

The experimental investigations herein are conducted using the social network application culled from the open source microservice benchmark, DeathStarBench [11]. This benchmark consists of a corpus of 36 different microservices that span three different service functionalities: *Compose-Post*, *UserTimeline*, and *HomeTimeline*.

A tiny Kubernetes cluster is meticulously assembled, consisting of four physical machines, each assigned to discrete operational roles: one for the master node, another for deploying stateful microservices, and the remaining two for deploying stateless microservices. All physical machines have identical hardware configurations, with 20 CPU cores and 32 GB of RAM. Notably, two of these machines, designated *N1* and *N2*, are dedicated to the deployment of stateless microservices, with each container having a resource footprint of 1 CPU core and 2GB of RAM.

Throughout the experimental campaign, an aggregate query rate of 800 queries per second (QPS) is generated and directed to the application. Notably, one-fourth of this massive query influx is directed to the *HomeTimeline* service, while the remainder is directed to the *UserTimeline* service. It is important to note that this query rate reflects the resource consumption patterns observed in production cluster environments.

7.2. Joint Optimization Policies

The instantiated joint optimization involves a two-step approach. First, it involves the judicious assignment of each container to an appropriate node within the cluster. Subsequently, it requires fine-tuning the workload distribution between UM and DM to achieve optimal results. In the context of this research, the *UserTimeline* (UT) and *HomeTimeline* (HT) services are used as illustrative examples. The topological relationship between UT and HT, highlighted in Fig. 12a, shows that UT and HT have distinct UM components, denoted as *U* and *H*, respectively, while sharing a common DM denotation as *P*. The workloads from *U* to *P* and from *H* to *P* are characterized by query rates of 600 QPS and 200 QPS, respectively. It should be emphasized that all container instances are deployed exclusively on either *N1* or *N2*, so the primary objective of this study is to determine the most efficient allocation scheme and workload configuration.

In our investigation, we consider the use of three different policies:

- **Fully Randomized**: Randomly assign containers to nodes, evenly distribute workload to containers.
- **Random Assignment, Workload Aware**: Randomly assign containers to nodes, distribute workload to intra-node containers first.
- **Assignment Aware, Workload Aware**: Strategically packing containers on same nodes, distribute workload to intra-node containers first.

A scenario resulting from the **Fully Randomized** policy is shown in Fig. 12b. As can be seen in the figure, containers are randomly assigned to either *N1* or *N2*, and the UM workload is evenly distributed across all containers of the DM *P*. To visually distinguish between intra-node and inter-node workloads, we use black lines to denote the former and red lines to denote the latter. Notably, this particular optimization policy results in a cumulative inter-node query rate of 400 QPS.

The worst scheduling result of the **Random Assignment, Workload aware** policy is shown in Fig. 12c. The figure shows that this scheduling result still randomly assigns containers to *N1* and *N2*, but it controls the workload between UM and DM containers. All workload from UM *U*

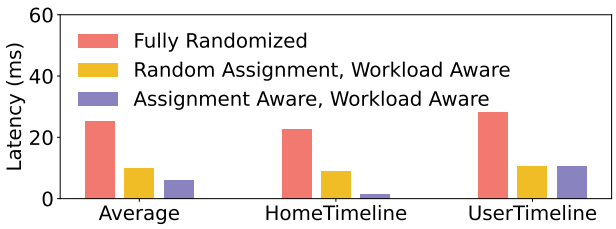


Figure 13: E2E latency of HomeTimeline and UserTimeline (95^{th} percentile).

is limited to containers *P3* and *P4*, which are on the same node of *U1* and *U2*. Therefore, the UM *U* doesn't send any inter-node workload, resulting in the total inter-node workload being reduced to 200 QPS.

Finally, the **Assignment Aware, Workload Aware** policy provides the optimal solution to this example. By assigning *H1* (*H2*), *U1* (*U2*), *P1* (*P3*), and *P2* (*P4*) to the same node, with limiting the workload to the same node containers, this policy achieves absolutely no inter-node workload.

The experimental results for the three different policies are shown in Fig. 13. As can be seen in the figure, the **Assignment Aware, Workload Aware** policy shows a significant reduction in E2E latency (95^{th} percentile) of over $4\times$ and $1.6\times$ compared to the **Fully Randomized** and **Random Assignment, Workload Aware** policies, respectively. A comparative analysis between the **Random Assignment, Workload Aware** and **Fully Randomized** policies reveals that the mere adoption of workload control results in a significant $2.6\times$ reduction in E2E latency (95^{th} percentile).

In summary, the formulation of a joint optimization aimed at minimizing inter-node communications has emerged as a key factor in improving container performance. However, achieving optimal results requires a meticulous examination of the microservice call graph and the resolution of container co-location challenges, underscoring the complexity inherent in modeling. Nevertheless, the concept of a joint optimization remains a promising avenue for improving performance and merits further exploration.

8. Related Work

Microservice benchmarks. Previous research has employed benchmarks such as Ditto [15], μqSim [32], and DeathStarBench [11] for the purpose of simulating microservices' behavior through diverse methodologies. In particular, Gan *et al.* introduced an automated framework designed to replicate end-to-end cloud services, enabling the representation of their behavior across hardware, I/O, networking layers, and the operating system [15]. Conversely, Zhang *et al.* presented a scalable simulation approach focused on capturing the interrelationships between microservices and their associated application semantics. This approach necessitates users to furnish dependency graphs and a description of the available server platforms [32]. Furthermore, Gan *et al.* conducted an empirical investigation comparing microservices and monolithic applications across various domains, utilizing the DeathStarBench benchmark as a foundational element [11].

Inter-node communication analysis. Gan *et al.* had demonstrate that network processing time may cost 36.3% of response time in DeathStarBench, they also improved network processing speed by $10-68\times$ with a FPGA to offload the entire TCP stack [11]. Shutian *et al.* in addition provides the avidance of co-location can reduce the response time by 22% on average, in some extreme cases, this improvement can be as high as 80% [18].

Cloud trace analysis. A plethora of studies have been conducted on the examination of cloud workloads [7], [10], [16], [17], [18], [19], [24], [26], [28], [29]. Reiss *et al.* delved into the heterogeneity and dynamism of the workload in the Google cluster. A comprehensive analysis of workloads in the Alibaba cluster was carried out by Lu *et al.* [17], where they explored the equilibrium between resource utilization, performance, and scheduling scalability. Shahrad *et al.* initiated a study on the characteristics of FaaS workload of Azure Functions, particularly their invocation frequencies [31]. However, these studies either do not cater to microservice architecture or lack detailed microservice information. Existing analyses of microservice trace [18] primarily concentrate on graph dependencies between different microservices and subsequently construct a stochastic model to simulate call graphs, based on a unique classification of microservice types. Despite this, these studies do not offer an extensive analysis concerning the performance and co-location of microservices.

9. Conclusion

Previous research in the field has predominantly analyzed microservices using benchmarks and simulators [11], [27], [30], [35] in small clusters or focused on graph dependencies [19]. However, to the best of our knowledge, this study represents the first comprehensive analysis of imbalances in microservices deployed in large production clusters. Our analysis delves into the imbalances in both resource utilization and RT performance, providing a profound understanding of the underlying mechanisms.

Through our investigation, we have discovered that container co-location and load-balancing policies significantly impact RT performance. Building upon these findings, we propose a novel joint optimization that aims to reduce inter-node communications, ultimately leading to significantly improved E2E performance. Our findings open up new possibilities for enhancing the scheduling and load-balancing policies of large-scale microservice systems, benefiting system providers by reducing costs and end users by improving performance.

It is important to note, however, that our joint optimization does not present an exact mathematical model. Therefore, it would be worthwhile to develop such a model to achieve global optimization.

Acknowledgments

We sincerely thank the anonymous reviewers of ISPA'23 for their valuable suggestions and comments. This work is supported in part by the Science and Technology Development Fund of Macau (0024/2022/A1), the Multi-Year Research Grant of University of Macau (MYRG2022-00119-FST), the Start-up Research Grant of University of Macau (SRG2021-00004-FST).

References

[1] "Alibaba cloud," https://www.alibabacloud.com/, 2023.

[2] "Alibaba cloud product," https://www.alibabacloud.com/product, 2023.

[3] "Alibaba trace," https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2022, 2023.

[4] "Amazon web services," https://aws.amazon.com/, 2023.

[5] "Cncf," https://www.cncf.io/, 2023.

[6] "Google cloud," https://cloud.google.com/, 2023.

[7] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben, "On the diversity of cluster workloads and its impact on research results," in *Proceedings of USENIX ATC*, 2018.

[8] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, May 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7540

[9] S. Chen, C. Delimitrou, and J. F. Martínez, "Parties: Qos-aware resource partitioning for multiple interactive services," in *Proceedings of ASPLOS*. ACM, 2019, pp. 107–120.

[10] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of ACM SOSP*, 2017, pp. 153–167.

[11] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of ASPLOS*. ACM, 2019, pp. 3–18.

[12] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou, "Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices," in *Proceedings of ASPLOS*, 2019, pp. 19–33.

[13] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, and J. Mars, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in *Proceedings of Eurosys*, 2019.

[14] M. Liang, Y. Gan, Y. Li, C. Torres, A. Dhanotia, M. Ketkar, and C. Delimitrou, "Ditto: End-to-end application cloning for networked cloud services," in *Proceedings of ASPLOS*, 2023.

[15] ——, "Ditto: End-to-end application cloning for networked cloud services," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 222–236. [Online]. Available: https://doi.org/10.1145/3575693.3575751

[16] Q. Liu and Z. Yu, "The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from alibaba trace," in *Proceedings of ACM SoCC*, 2018, pp. 347–360.

[17] C. Lu, H. Xu, K. Ye, G. Xu, L. Zhang, G. Yang, and C. Xu, "Understanding and optimizing workloads for unified resource management in large cloud platforms," in *Proceedings of EuroSys*, 2023.

[18] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *Proceedings of ACM SoCC*, 2021.

[19] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, J. He, and C.-Z. Xu, "An in-depth study of microservice call graph and runtime performance," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[20] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, J. He, G. Yang, and C. Xu, "Erms: Efficient resource management for shared microservices with sla guarantees," in *Proceedings of ASPLOS*, 2023.

[21] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, G. Yang, and C. Xu, "The power of prediction: Microservice auto scaling via workload learning," in *Proceedings of the 13th Symposium on Cloud Computing*, ser. SoCC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 355–369. [Online]. Available: https://doi.org/10.1145/3542929.3563477

[22] J. Park, B. Choi, C. Lee, and D. Han, "Graf: A graph neural network based proactive resource allocation framework for slo-oriented microservices," in *Proceedings of ACM CoNext*, 2021.

[23] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, "Firm: An intelligent fine-grained resource management framework for slo-oriented microservices," in *Proceedings of USENIX OSDI*, 2020.

[24] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of ACM SoCC*, 2012, pp. 1–13.

[25] K. Rzadca, P. Findeisen *et al.*, "Autopilot: workload autoscaling at google," in *Proceedings of EuroSys*, 2020.

[26] M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proceedings of USENIX ATC*, 2020, pp. 205–218.

[27] A. Sriraman and T. F. Wenisch, "μ suite: a benchmark suite for microservices," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 1–12.

[28] H. Tian, Y. Zheng, and W. Wang, "Characterizing and synthesizing task dependencies of data-parallel jobs in alibaba cloud," in *Proceedings of ACM SoCC*, 2019, pp. 139–151.

[29] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: the next generation," in *Proceedings of Eurosys*, 2020, pp. 1–14.

[30] T. Ueda, T. Nakaike, and M. Ohara, "Workload characterization for microservices," in *2016 IEEE international symposium on workload characterization (IISWC)*. IEEE, 2016, pp. 1–10.

[31] K. Veeraraghavan, J. Meza, D. Chou, W. Kim, S. Margulis, S. Michelson, R. Nishtala, D. Obenshain, D. Perelman, and Y. J. Song, "Kraken: leveraging live traffic tests to identify and resolve resource utilization bottlenecks in large scale web services," in *Proceedings of USENIX OSDI*, 2016, pp. 635–651.

[32] Y. Zhang, Y. Gan, and C. Delimitrou, "μqsim: Enabling accurate and scalable simulation for interactive microservices," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 212–222.

[33] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, "Sinan: Ml-based and qos-aware resource management for cloud microservices," in *Proceedings of ASPLOS*, 2021.

[34] L. Zhao, Y. Yang, K. Zhang, X. Zhou, T. Qiu, K. Li, and Y. Bao, "Rhythm: component-distinguishable workload deployment in datacenters," in *Proceedings of EuroSys*, 2020.

[35] X. Zhou, X. Peng, T. Xie, J. Sun, C. Xu, C. Ji, and W. Zhao, "Benchmarking microservice systems for software engineering research," in *Proceedings of ICSE*, 2018.