Priority-Based Optimization of I/O Isolation for Hybrid Deployed Services

Jiancheng Zhang[1,2], Youhuizi Li[1,2](✉), Li Zhou[1,2], Zujie Ren[3], Jian Wan[1,4], and Yuan Wang[5]

[1] Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, Hangzhou Dianzi University, Hangzhou, China
{huizi,zhouli}@hdu.edu.cn

[2] School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China

[3] Zhejiang Lab, Hangzhou, China
renzj@hdu.edu.cn

[4] School of Information and Electronic Engineering, Zhejiang University of Science and Technology, Hangzhou, China
wanjian@zust.edu.cn

[5] Key Enterprise Research Institute of NetEase Big Data of Zhejiang Province, Netease Hangzhou, Network Co.Ltd, Hangzhou, China

Abstract. With the increasing of software complexity and user demands, collaborative service is becoming more and more popular. Each service focuses on its own specialty, their cooperation can support complicated task with high efficiency. To improve the resources utilization, virtualization technology like container is used and it enables multiple services running in the same physical machine. However, since the host physical machine is shared by several services, the resource competition is inevitable. Isolation is an effective solution, but the weak isolation mechanisms of container cannot handle such complicated scenarios. In the worst situation, the performance of services cannot meet the requirements and the system may crash. In order to solve this problem, we propose a priority-based optimization mechanism for I/O isolation after analyzing the characteristics of typical service workloads. Based on the real-time performance data, priority is automatically assigned to each service and corresponding optimization methods are applied. We evaluate the optimization effects of the priority-based mechanism in both static and dynamic workload cases, besides, the influence of different priority order is also analyzed. The experimental results show that our approach can indeed improve the system performance and guarantee the requirements of all the running services are satisfied.

Supported in part by the Natural Science Foundation of Zhejiang Province under Grant LQ18F020003 and Grant LY18F020014, and in part by the Natural Science Foundation of China under Grant 61802093 and Grant 61572163, in part by the Xi'an Key Laboratory of Mobile Edge Computing and Security (201805052-ZD3CG36) and in part by the Key Research and Development Program of Zhejiang Province under Grant 2018C01098.

X. Wang et al. (Eds.): CollaborateCom 2019, LNICST 292, pp. 28–44, 2019.
https://doi.org/10.1007/978-3-030-30146-0_3

Keywords: Priority-based · I/O isolation · Container · Hybrid deployment

1 Introduction

With the development of the Internet and computing technology, one service cannot support the various requirements of users, especially when applications are becoming more and more complex. For example, the web services nowadays are often composed of multiple sub-services such as location, video and communication to satisfy users' demands [5]. The cost of developing and maintaining a "perfect" application/service that has all the functions that users request will be huge. Hence, collaborative service is applied [18,21]. Each service is responsible for a small portion, the workload is decomposed and distributed to several services. By cooperating with each other and sharing intermediate data and resources, the complex task can be finished efficiently. Single service normally is simple and the resources requirement is low. To improve the resources utilization of servers, multiple services are deployed on a same physical machine. If these services are working on a same task, they can use memory/disk to transmit the data instead of network, which also further improves the service performance and security.

Virtualization is used to support multiple services running in one physical server. Container [1,4], as a lightweight virtualization technology which does not pack the guest operating system, is widely used since it provides high resources utilization and low overhead. Although we deploy multiple services in one machine, the services themselves are independent, even they may work together. To make sure their execution are normal and not influenced by others, there should be strict resource isolation mechanisms. However, container mainly relies on the original Namespace and the Cgroups of the Linux to provide isolation feature [16,20]. It cannot properly handle the severe competition situations. Besides, the workload and functionality of collaborative services are various, the different behavior/requirements of the services make the resource competition even more complicated. The worst case is that the requirements of most services cannot be satisfied and the system will crash. Hence, there is a great challenge to cope with the resource isolation and performance optimization for hybrid deployed services.

Existing isolation optimization methods mainly target homogeneous deployment scenario, that is, the deployed services have the same type of performance requirements. So, single optimization method is enough to protect all services. For latency-sensitive services, the commonly used approaches are sending requests repeatedly and rate limiting [3,9,12,17,19,22,23]. For throughput-first services, disk allocation and I/O concurrency control are applied [2,6–8,10]. Asides from one-dimensional (latency or throughput) performance restriction, there are also services that require both metrics to meet the standards. The optimization method [11] is more conservative, and the resources utilization is relatively low. This two-dimensional services situation still belongs to homogeneous case since all the services are the same type. Hence, the aforementioned solutions can only handle specific type of services, and they are not appropriate for hybrid deployment scenario. In addition, our previous work PINE [13] can cope with one latency-sensitive service plus multiple throughput-first services scenario. It classifies services according to their performance indicators and applies different optimization methods accordingly. By leveraging the idle server resource of a latency-sensitive service (when its workload is light), the enterprise can support other throughput-first services to make extra profits. However, it cannot be guaranteed that PINE also works for multiple latency-sensitive services and multiple throughput services scenario which usually happens in collaborate services.

In this paper, we extend PINE and propose an priority-based I/O isolation optimization mechanism which targets more general hybrid deployed scenarios. First, according to the services characteristics, different optimization methods are applied. For examples, adjusting the I/O concurrency level for latency-sensitive services, and modifying disk allocation for throughput-first services. Then, to support the execution of multiple latency-sensitive services, a prioritization algorithm is developed. The latency-sensitive services are sorted based on their influence to the whole system, and the optimization method is applied accordingly to maintain the status. As far as we know, this optimization mechanism is the first method that effectively handles hybrid deployment scenario and ensures that each service can meet its requirements.

The rest of this paper is organized as follows. Section 2 reviews the related work in the performance optimization field. Section 3 analyzes the characteristics of the hybrid deployment scenario and illustrates the priority-based optimization mechanism. Section 4 comprehensively evaluates the optimization effects and the performance of the prioritization algorithm. Section 5 summarizes the paper and describes the future work.

2 Related Work

Performance optimization is a popular research topic in recent years, especially with the widely-spread virtualization technology. According to the type of the deployed services, previous researches can be divided into the following three categories: one-dimensional homogeneous services, two-dimensional homogeneous services, and single hybrid heterogeneous services.

One-Dimensional Homogeneous Services Scenario: The most common types of performance indicators are 99.9th percentile latency and throughput. The services in this scenario have either latency or throughput as their requirements. For latency-sensitive services, there are three optimization method: (1) Modifying the queue scheduling strategy of the Linux kernel. Li *et al.* [12] believed that the FIFO is a more friendly scheduling strategy for 99.9th percentile latency. (2) Sending requests redundantly to reduce the blocking possibility. Google [3] proposed that the same request can be sent redundantly, and the fastest response will be take. The latency is improved as a result of resource overuse. (3) Integrated scheduling. Wang *et al.* [19] designed Cake, a multi-layer optimization framework, to efficiently schedule several resources together so the performance can be improved. For throughput-first services, disk resource allocation is the commonly used optimization approach. Gulati *et al.* [8] proposed mClock, which sets the upper and lower disk threshold based on the service requirements in advance, to control the disk resource. In one-dimensional homogeneous services scenario, the optimization methods only work for single type of services (latency or throughput), it cannot handle hybrid deployment scenario and will inevitably leads to part of the services failed.

Two-Dimensional Homogeneous Services Scenarios: All the services' type in this scenario are also the same, but each service can contain two performance indicators (latency and throughput) instead of one. The PSLO framework [11] exactly targets this situation. It satisfies the latency and throughput requirements of each service by controlling the I/O rate and concurrency level. However, since there is an obvious trade-off between latency and throughput in some cases, PSLO provides a boundary curve which describes this relationship. If the resource competition is too fierce that exceeds the boundary curve, PSLO will not take any action. Besides, the optimization strategy of PSLO is also conservative, and the resource utilization is relatively low. Hence, the optimization method used in this scenario cannot cope with hybrid deployment either.

Single Hybrid Heterogeneous Services Scenario: In this scenario, services with different requirements are deployed together, specifically, only one service can have a "special" type of requirements that different with all others. For example, one latency-sensitive service with multiple throughput-first services. PINE [13] is developed to handle this scenario, it achieves latency optimization through adjusting I/O concurrency level and throughput optimization using disk allocation. As a result, all services can satisfy their performance requirements. However, PINE has over-optimization problem and resource utilization degradation issue when multiple latency-sensitive services exist.

After analyzing the existing related works, we plan to propose an optimization mechanism which ultimate goal is to efficiently handle general multiple hybrid deployment situations and guarantee all the services' requirements can be satisfied as much as possible.

3 Priority-Based Optimization Mechanism

In this section, we first discuss the general service types in the hybrid deployment scenario, then the performance interference and existing optimization methods are analyzed. Following that, we illustrate the prioritization algorithm and priority-based optimization mechanism.

3.1 Service Type

After analyzing modern collaborative services, we summarized mainly two types of services: latency-sensitive service and throughput-first service.

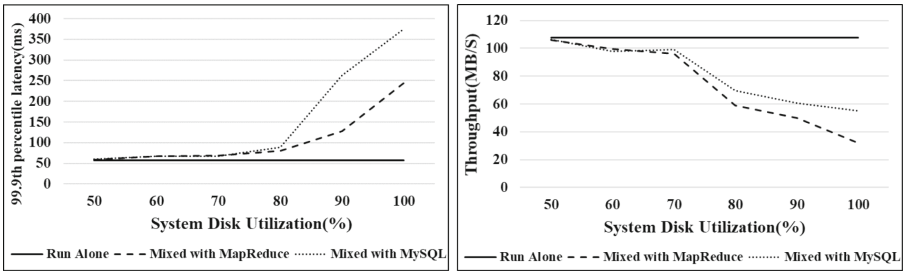
Latency-Sensitive Service: With the developing of Internet, the response time becomes a critical data that directly influence the user experience, especially for social media service, search engines, online maps [14]. To improve the performance, services normally will split the request to several sub-requests and execute in a parallel way. So, the response time is defined by the longest task. Compare with the average latency, the 99.9th percentile latency is selected as the performance metric for latency-sensitive services. The execution of latency-sensitive services are usually thread-driven or event-driven. Thread-driven services use synchronous blocking I/O and generate a new thread for each new user's I/O request, while event-driven services apply asynchronous non-blocking I/O and handle all I/O requests by several worker threads. The I/O processing speed of a server is commonly fast until there are many request pending. The latency will be amplified in the 99.9th percentile metric, even the interference is very small, which leads to latency performance violation.

Throughput-First Service: Services that need to process batch jobs pay more attention to throughput, such as analyzing working logs [15]. These services can be delayed occasionally or restarted over time. The throughput is the main metric that user cares most, which is decided by the available disk bandwidth. When the disk resource of a service is enough, it can achieve the reasonable performance requirements. If the resource competition is severe, other services may malicious occupy the shared disk, which leads to the failure of the throughput-first service.

3.2 Performance Interference in Hybrid Deployment Scenario

In real usage scenarios, services that deployed on a same physical machine are randomly selected. We target at general hybrid deployed situations where several types of services may mix together and each type also can have multiple services running. Hence, the interference exists in different types of services, and the resource competition also comes from other same-type services. To analyze the performance influence, we did the following experiments. MySQL represents latency-sensitive service, which enables 64 threads to read and write 10 tables together (the initial IO concurrency level is 64); Hadoop MapReduce as the throughput-first service executes the MapReduce operation for 100 files.

Latency-Sensitive Service: Figure 1(a) shows the latency comparison under three cases: running alone, mixed with MapReduce (different service type) and mixed with MySQL (same service type). In the mixed with MapReduce case, the workload of MySQL is constant and the workload of MapReduce gradually increases. Similarly, in the mixed with MySQL case, the workload of one MySQL service increases. The results show that there is no obvious difference of the three cases when the disk bandwidth usage is low (e.g. 50%, 60%, and 70%), which means the competition of disk resource is not intense. While the disk usage rate rises to 80% or more, there is a great delay in mixed cases. The 99.9th percentile latency of MySQL in the mixed with MapReduce case is nearly 6 times higher than running alone, and it is 8 times in the mixed with MySQL case when the disk usage was close to 100%. Hence, the loss of performance will be significant



(a) Performance comparison of latency-sensitive services.

(b) Performance comparison of throughput-first services.

Fig. 1. Performance interference in hybrid deployment scenario.

when the disk usage reaches a threshold (like 80% in this experiment), and the impact of the same-type services is greater than different types of services for latency-sensitive services.

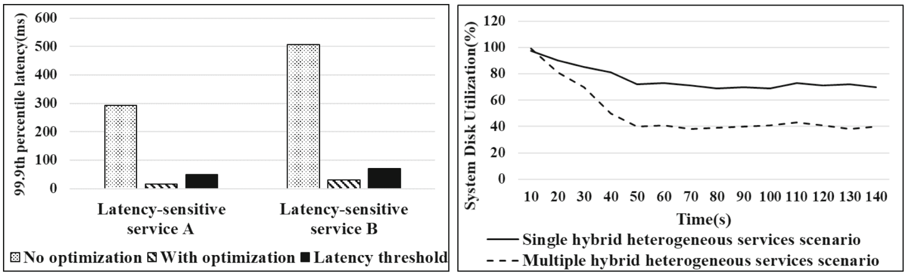
Throughput-First Service: The configurations are same with the previous experiments except that the position of MapReduce and MySQL is interchanged, and we focused on the throughput of MapReduce. As the Fig. 1(b) presented, the performance trend of MapReduce was similar with previous MySQL's. The influence of disk resource competition became significant when the disk utilization is around 70%, and the impact of the same type of service is also greater.

The two experiments show that the impact of disk resource competition on service performance will suddenly increase after the system disk utilization reaches a certain threshold. Both latency-sensitive services and throughput-first services will be interfered by the same-type and different types of services, and the influence from same-type services is more serious.

3.3 Existing Isolation Method

To handle the single hybrid heterogeneous services scenario, PINE applies I/O concurrency control for latency-sensitive services and disk resource allocation for throughput-first services. In this section, we first analyze whether the above two optimization methods are still feasible in general hybrid deployed scenarios (multiple hybrid heterogeneous services scenario). Since the limitation of the I/O concurrency control method is illustrated in [13], we focus on the disk allocation method and the combination of the two methods.

Disk Allocation: Docker creates virtual device and assigns distinct device number for each container, so Cgroups can be used to implement disk allocation method. Based on the throughput requirements of the services, we can perform an overall disk resource partition at the operating system level. Throughput-first services can get their own share, which decreases the performance interference caused by the same-type services. For the latency-sensitive services, Cgroups cannot accurately allocate disk for each individual service since they do not have throughput performance value. From the perspective of container, the influence of single latency-sensitive service and multiple latency-sensitive services have no difference as long as the allocated disk is fixed. So the disk resource is allocated for all latency-sensitive services instead of individual latency-sensitive service. Hence, Cgroups can effectively guarantee the performance of throughput-first services in general hybird deployment scenario.



(a) Comparison of 99.9th percentile latency. (b) Comparison of system disk utilization.

Fig. 2. Optimization comparison in hybrid deployed scenarios.

Combination of I/O Concurrency Control and Disk Allocation: After allocating disk resource using Cgroups for each throughput-first service and the group of all latency-sensitive services, the I/O concurrency control algorithm in PINE is applied to each latency-sensitive service. There are two latency-sensitive services (A and B) in the experiment, the optimization comparison results are shown in Fig. 2(a). Comparing with the no optimization case, the I/O concurrency control method successfully decreased the latency of service A and B. After several iteration intervals, the 99.9th percentile latency of service A and B are around 16 ms and 31 ms respectively, which is much lower than the required latency value. However, with the same experimental configuration, the latency value in the single hybrid heterogeneous services scenario is close to its standard latency requirement. To figure out the behind reason, we analyzed the system disk utilization in the two scenarios. Figure 2(b) shows that the disk utilization decreases to a stable value after applying the optimization method in both cases. In the single hybrid heterogeneous services scenario, the disk utilization dropped to around 70%, which is 30% more than the multiple hybrid heterogeneous services scenario. The results indicate that PINE sacrifices the resource utilization to guarantee the latency. But it over-optimized, the latency is less than 50% of the required value. Hence, this method is not appropriate for general hybrid deployed scenarios.

Analysis: In general hybrid deployed scenarios, there are two levels of resource competition: different-type service competition and same-type service. For throughput-first services, their performance directly related to available disk resource. After given fixed disk space, they are total isolated from others (including other throughput-first services and latency-sensitive services) and their performance is also determined. For latency-sensitive services, there is no clear mathematical relationship of I/O concurrency and 99.9th percentile latency. Besides, the allocated disk resource is for all latency-sensitive services. Since the behavior of same-type services are very similar, the resource competition is more intense in this case. As a result, the existing method cannot efficiently handle the same-type service competition of the latency-sensitive services. The I/O concurrency limitation is for all latency-sensitive services, the difference between each service is not considered. So, the over-optimized situation happens. The potential solution is control I/O concurrency of each service one by one. Assuming there are three latency-sensitive services: A, B and C. After we control the I/O concurrency of A, the system needs sometime to react and it also influences the behavior/performance of service B and C. Hence, to eliminate the over-optimization, we can only modify one variable in each iteration and wait for the effects, then decide the next move.

3.4 Prioritization Algorithm

To deal with same-type service competition and over-optimization phenomenon of latency-sensitive services, the asynchronous I/O concurrency control is proposed. If the disk resources of one service changed, it will also influence other services' performance, while the assumption of synchronous I/O concurrency control is the environment keeps constant for all services within the iteration interval. Hence, asynchronous I/O concurrency control method, which considers the interactions of latency-sensitive services and only sets the I/O concurrency level of one service in each iteration, can effectively decrease the optimization time and improve the performance as well as resource utilization.

Prioritization algorithm is designed to automatically select the service whose I/O concurrency should be updated at each iteration interval. The throughput variation after applying the concurrency control is the main factor we consider. If the throughput of a service changes greatly, it will have a huge impact on the shared disk resource, which further influences other latency-sensitive services. Therefore, the idea of the proposed algorithm is that the greater the impact on the disk resource, the higher the priority should be. If concurrency control is applied to small-impact services first, then when it comes to control large-impact services, the latest configuration will total sacrifice the optimized results of small-impact services. Hence, we will adjust the I/O concurrency of the highest priority service, that is, the service whose throughput variation is maximum.

The service throughput variation is decided by the current throughput and the violation degree of 99.9th percentile latency. If the violation degree is high, then the I/O concurrency level control will be stronger, the variation rate of the throughput will be high. Combined with the current throughput value, the throughput variation can be calculated. Assuming L_{cur} represents the current

99.9th percentile latency and L_{SLO} represents the required 99.9th percentile latency, then the violation degree Vio is defined as:

$$Vio = \frac{L_{cur} - L_{SLO}}{L_{SLO}} \tag{1}$$

With the current throughput Th_{cur}, the throughput variation Th_{dif} can be calculated as:

$$Th_{dif} = \frac{Vio}{L_{Now}/L_{SLO}} * Th_{cur} \tag{2}$$

3.5 Priority-Based Optimization Mechanism

Based on the negative feedback regulation, priority-based optimization mechanism collects the performance requirements and real-time data of all running services, then applies the customized optimization strategy according to the difference. The architecture is shown in Fig. 3. The optimization process includes throughput optimization and 99.9th percentile latency optimization. Since it takes some time for the optimization to take effect in the system, the iteration interval of data collection and optimization is set to 10 s based on practical experience.

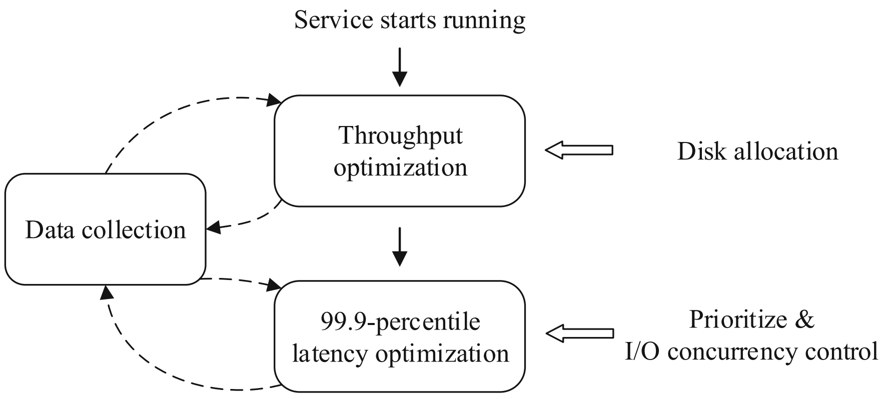
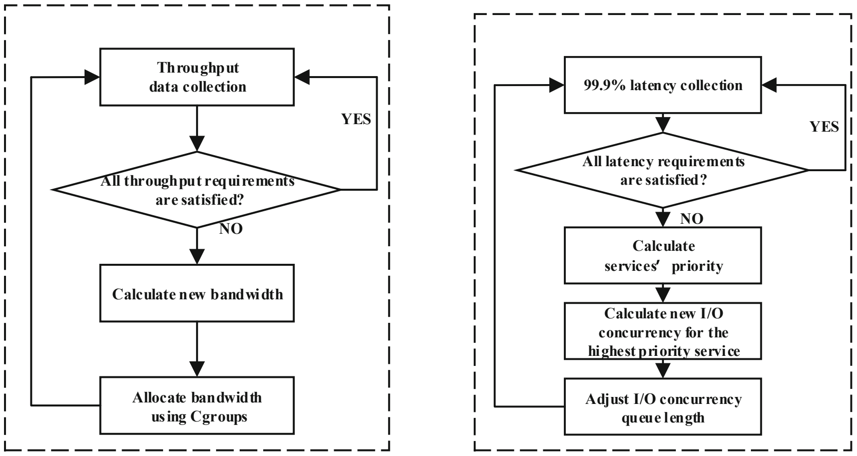


Fig. 3. The architecture of priority-based optimization mechanism.

Throughput-First Services: The performance requirement for this type of services normally is to maintain the throughput above the predefined threshold. As the Fig. 4(a) illustrates, the first step is collecting throughput data. It is necessary to distinguish the different services' traffic since the processes in the container are all running on the host machine. Docker builds a virtual disk volume with an distinct device ID for each container. So, the IOSTAT tool can be used to get the performance data (e.g. IOPS) of each service, and then the real-time throughput value can be calculated. If the throughput does not match the predefined threshold, the violation happens. To control the throughput, Cgroups is leveraged to allocate disk resources for each throughput-first service based on their thresholds.


Throughput data collection
All throughput requirements are satisfied?
YES
NO
Calculate new bandwidth
Allocate bandwidth using Cgroups
99.9% latency collection
All latency requirements are satisfied?
YES
NO
Calculate services' priority
Calculate new I/O concurrency for the highest priority service
Adjust I/O concurrency queue length

(a) Throughput optimization. (b) 99.9th-percentile latency optimization.

Fig. 4. The optimization process.

Latency-Sensitive Services: Similar to throughput optimization, Fig. 4(b) describes the process of latency-sensitive services. To calculate the 99.9th percentile latency, the response time of all the requests of the service within the iteration interval is logged. The collected latency is sorted in ascending order, and the data at the 99.9th percentile position in the sequence is tagged as the 99.9th percentile latency of the service. Comparing with the latency threshold, if the violation exists, we need to modify the I/O concurrency level. First, the priority of each service which does not reach the latency requirements is calculated according to the proposed prioritization algorithm. The I/O concurrency level of the highest priority service should be adjusted. The new value of the I/O concurrency level for the next cycle is estimated using a linear fitting equation in a multi-iterate manner. To control the I/O concurrency level, the I/O concurrent queue length of the selected service is set to the calculated new value. If the current number of outstanding requests (i.e., being executed) is less than the queue length, the new request is allowed to enter the queue, otherwise the request is refused.

4 Evaluation

After presenting the experimental setup, we evaluate the optimization effect of the priority-base optimization mechanism and the influence of service priority.

4.1 Experimental Setup

The experimental setup is shown in Table 1, MySQL and Hadoop MapReduce represent 99.9th percentile latency-sensitive service and throughput-first service respectively. The Hadoop cluster is composed of three nodes, including one master and two slaves.

We target at general hybrid deployed scenarios, that is, there are several latency-sensitive services and multiple throughput-first services running together. Without generosity, we simulate three latency-sensitive services and three throughput-first services. The latency requirements are 60 ms (service L-A), 80 ms (service L-B), 100 ms (service L-C), and the throughput requirements are 20 MB/s (service T-A), 40 MB/s (service T-B), 60 MB/s (service T-C).

Table 1. Experimental setup.

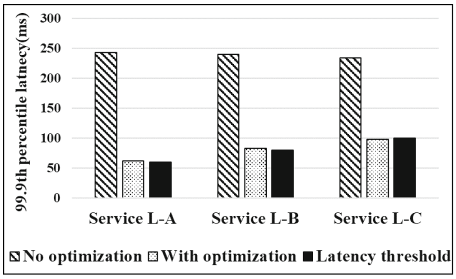
Item	Version
CPU	**16 AMD Opteron Processor 6136**
Memory	**32 GB**
Storage	**5,400 RPM 120 GB SATA disks**
Operating System	**CentOS7**
Linux Kernel	**3.10.5-3.el6.x86_64**
Docker	**1.17-ce**
MySQL	**MySQL 5.6**
Hadoop	**Apache Hadoop 1.0**

4.2 Optimization Evaluation

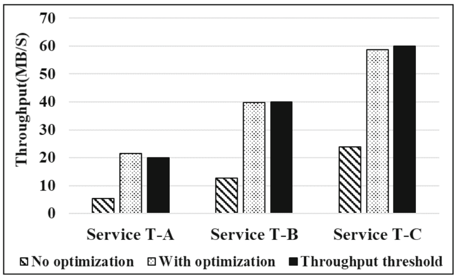
Constant Workload Optimization: All the services' workload are constant in this experiment. The workload of each service is equal to the load amount when the service runs alone and just satisfies its performance requirement. The comparison under the with and without optimization cases is shown in Fig. 5.

Latency-Sensitive Services: As Fig. 5(a) demonstrates, there are serious latency violations of the three latency-sensitive services in the no optimization case. The 99.9th percentile latency of service L-A has reached 243.13 ms, which was 305% worse than its latency threshold (60 ms). The similar situations for service L-B and L-C. On the contrary, the 99.9th percentile latency of the three services can be stabilized at the predefined latency threshold. Taking service L-B as an example, the 99.9th percentile latency was reduced by 65.5% compared to the no optimization case and successfully reached the 60 ms latency requirement.

Throughput-First Service: As Fig. 5(b) shows, there are also serious throughput violations of the three throughput-first services in the no optimization case. Taking service T-A (requirement is 20 MB/s) as an example, its throughput was only 5.42 MB/s, and it is 72.9% lower than the threshold. After applying the optimization, the throughput was stabilized at 21.52 MB/s.



(a) Latency-sensitive services.



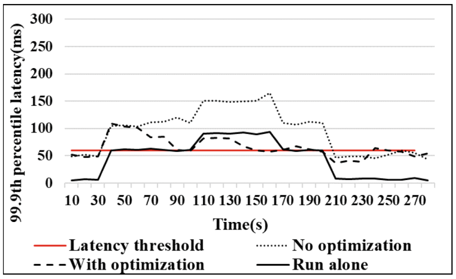
(b) Throughput-first services.

Fig. 5. The performance comparison under constant workload.

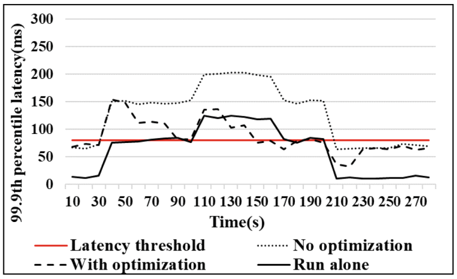
The experimental results indicate that the isolation optimization method proposed in this paper can indeed optimize the performance of constant workload services in hybrid deployed scenarios.

Dynamic Workload Optimization: To evaluate the priority-based optimization mechanism in the real-world usage scenarios, we pay more attention to dynamic workload optimization results. The workloads in actual production environment are commonly various with time, especially for latency-sensitive services. Without generosity, the workload changing of the same-type services is happened at the same time, and the transition from non-violation to violation will also occurs simultaneously. In this case, the resources competition is stronger than the workload changes one by one. Besides, based on the previous observation experiments, the performance influence from same-type services is greater than the effects of different-type services. Hence, we dynamically modify the workload of same-type services and the different-type services workload are constant for simplicity.

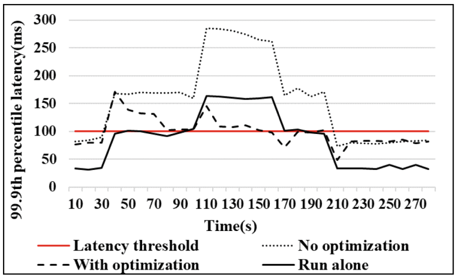
Latency-Sensitive Services: As can be seen from Fig. 6, there was no latency violation of the three MySQL services due to the low workload before the 30 s. Then the workload increased suddenly to make their 99.9th percentile latency over the thresholds. Take service L-A (threshold is 60 ms) as an example, its 99.9th percentile latency reaches 105.11 ms at the time 30 s. In the with optimization case, the latency dropped back to 60.82 ms after six iteration cycles. The workload increased again at 110 s, which generated another latency violation for all latency-sensitive services. In the with optimization case, the latency of the three services all fall back to the threshold after a few cycles. Hence, no matter how the workload changes, the latency can be kept near the predefined threshold, and even there is a violation, it can also be adjusted back within several iteration cycles. The priority-based optimization mechanism can efficiently cope with the latency-sensitive service isolation problem to guarantee their performance.



(a) Latency-sensitive service L-A.

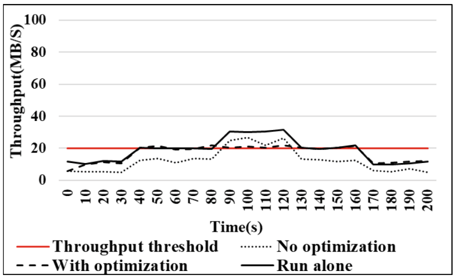


(b) Latency-sensitive service L-B.

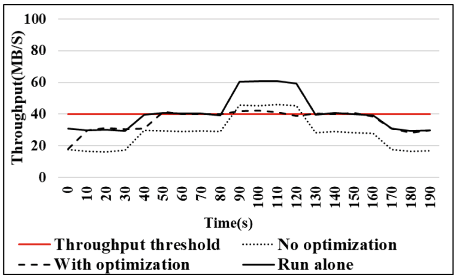


(c) Latency-sensitive service L-C.

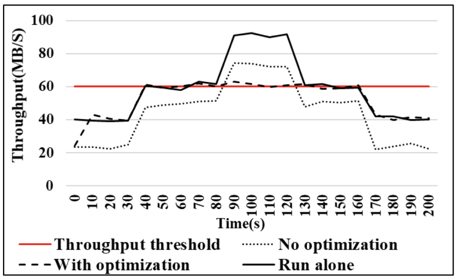
Fig. 6. Performance comparison of latency-sensitive services under dynamic workload.



(a) Throughput-first service T-A.



(b) Throughput-first service T-B.



(c) Throughput-first service T-C.

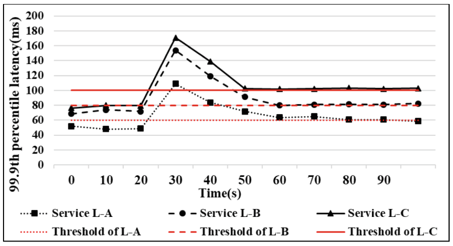
Fig. 7. Performance comparison of throughput-first services under dynamic workload.

Throughput-First Services: Similarly, we modified the workload of the three throughput-first services and keeps latency-sensitive services' constant. As illustrated in Fig. 7, take the service T-A as an example, its throughput was below the threshold due to the low workload in the beginning (before 30 s). Compared to the no optimization case, the throughput in the with optimization case was more close to the value when running alone. When the workload increased and the throughput was higher than the threshold, the optimization mechanism was applied to avoid excessive occupying the resource and the throughput was restricted around the predefined threshold. The experimental results show that the priority-based optimization mechanism can also effectively handle throughput-first services in the hybrid deployed scenarios.

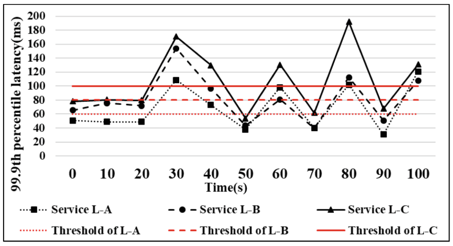
4.3 Priority Influence

To cope with latency-sensitive same-type service competition problem and eliminate over-optimization, we propose prioritization algorithm which assigns priority to each latency-sensitive service based on their impact to the system disk usage and the highest service is selected to apply I/O concurrency control. In this part, we evaluate the influence of different priority sequence on the system performance. Since the priority is calculated for each latency-sensitive service, the experimental configuration is similar to the dynamic workload latency-sensitive services case.

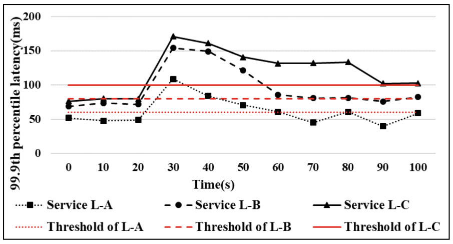
Figure 8 show the performance comparison, we analyzed three cases: Fig. 8(a) uses the order calculated from the proposed prioritization algorithm, Fig. 8(b) does not consider the order and applies I/O control for all latency-services at the same time, and Fig. 8(c) uses the reverse order of Fig. 8(a)'s. In Fig. 8(a), the workload is increased in the 30th second, and the 99.9th percentile latency of the three services took only 2–3 cycles from violation state to normal state. While in Fig. 8(b), it did not consider impact from other same-type services and optimized all three services at the same time, which lead to over-optimization. In order to relief from over-optimization, the system increased the I/O concurrency level. Unfortunately, this action only causes the latency violation in the next cycle. The back and forth process repeatedly happen, and the optimization time is obviously extended. In Fig. 8(c), we actually first controlled the smallest impact latency-sensitive service, service L-A. However, since the impact of service L-A is small, when the system comes to control high impact service, the previous results of modifying service L-A may be affected and service L-A need to be re-optimized again in the subsequent optimization cycle. As a result, although the 99.9th percentile latency of the three latency-sensitive services tends to be stable and reached their respective latency threshold, the optimization cycle is longer compared with Fig. 8(a), and its ability to handle burst high workloads is even worse.



(a) Prioritization algorithm order (descending order).



(b) Simultaneously.



(c) The reverse order (ascending order) .

Fig. 8. The latency comparison under three different priority orders.

In summary, optimizing one latency-sensitive service in each iteration cycle can greatly avoid over-optimization. Besides, applying the optimization based on the impact descending order (calculated according to the prioritization algorithm), the system can effectively decrease the optimization time.

5 Conclusion

To improve the performance of hybrid deployed collaborative services, we focus on the I/O isolation optimization problem. Firstly, we abstract the typical hybrid deployment scenarios by analyzing the execution process and potential interference of collaborative services. Then, we propose a priority-based isolation mechanism, which automatically assigns priority based on the real-time performance data and applies appropriate optimization methods. Comparing with the no-optimization case, for the latency-sensitive services, the 99.9th percentile latency violation can be recovered to the normal value in one or two cycles with a decreasing of 70%; for the throughput-first services, the throughput can achieve 50% higher in one cycle. The experimental results show that the priority-based optimization mechanism can effectively guarantee the performance of hybrid deployed services.

In the future, we will further improve the proposed mechanism from the aspects of more complex usage scenarios, more types of sub-services, optimal priority order and less recovery time.

References

1. Bernstein, D.: Containers and cloud: from LXC to docker to kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2015)
2. Bruno, J., Brustoloni, J., Gabber, E., Mcshea, M., Silberschatz, A.: Disk scheduling with quality of service guarantees. In: IEEE International Conference on Multimedia Computing & Systems (1999)
3. Dean, J., Barroso, L.A.: The tail at scale. Commun. ACM **56**(2), 74–80 (2013)
4. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and linux containers. In: IEEE International Symposium on Performance Analysis of Systems & Software (2007)
5. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Model-based verification of web service compositions. In: 18th IEEE International Conference on Automated Software Engineering (ASE 2003), Montreal, Canada, pp. 152–163, 6–10 October 2003. https://doi.org/10.1109/ASE.2003.1240303
6. Gulati, A., Ahmad, I., Waldspurger, C.A.: Parda: proportional allocation of resources for distributed storage access. In: Proceedings of the Conference on File & Storage Technologies (2009)
7. Gulati, A., Shanmuganathan, G., Zhang, X., Varman, P.: Demand based hierarchical QOS using storage resource pools. In: Usenix Conference on Technical Conference (2012)
8. Gulati, A., Varman, P.J.: mClock: handling throughput variability for hypervisor IO scheduling. In: Usenix Conference on Operating Systems Design & Implementation (2011)
9. Jeon, M., et al.: Predictive parallelization: taming tail latencies in web search (2014)
10. Jin, W., Chase, J.S., Kaur, J.: Interposed proportional sharing for a storage service utility. ACM Sigmetrics Perform. Eval. Rev. **32**(1), 37–48 (2004)
11. Li, N., Jiang, H., Feng, D., Shi, Z.: PSLO: enforcing the X^{th} percentile latency and throughput slos for consolidated VM storage. In: Proceedings of the Eleventh European Conference on Computer Systems, EuroSys 2016, London, United Kingdom, pp. 28:1–28:14, 18–21 April 2016. https://doi.org/10.1145/2901318.2901330
12. Li, N., Jiang, H., Feng, D., Shi, Z.: Customizable slo and its near-precise enforcement for storage bandwidth. ACM Trans. Storage **13**(1), 6 (2017)
13. Li, Y., Zhang, J., Jiang, C., Wan, J., Ren, Z.: Pine: optimizing performance isolation in container environments. IEEE Access **7**, 30410–30422 (2019)
14. Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P.: Heracles: improving resource efficiency at scale. ACM Sigarch Comput. Archit. News **43**(3), 450–462 (2015)
15. Marshall, P., Keahey, K., Freeman, T.: Improving utilization of infrastructure clouds. In: IEEE/ACM International Symposium on Cluster (2011)
16. McDaniel, S., Herbein, S., Taufer, M.: A two-tiered approach to I/O quality of service in docker containers. In: 2015 IEEE International Conference on Cluster Computing, CLUSTER 2015, Chicago, IL, USA, pp. 490–491, 8–11 September 2015. https://doi.org/10.1109/CLUSTER.2015.77
17. Suresh, L., Canini, M., Schmid, S., Feldmann, A.: C3: cutting tail latency in cloud data stores via adaptive replica selection. In: Usenix Conference on Networked Systems Design & Implementation (2015)
18. Touzi, J., Benaben, F., Pingaud, H., Lorré, J.P.: A model-driven approach for collaborative service-oriented architecture design. Int. J. Prod. Econ. **121**(1), 5–20 (2009)

19. Wang, A., Venkataraman, S., Alspaugh, S., Katz, R., Stoica, I.: Cake: enabling high-level SLOs on shared storage systems. In: ACM Symposium on Cloud Computing (2012)
20. Xavier, M.G., Oliveira, I.C.D., Rossi, F.D., Passos, R.D.D., Matteussi, K.J., Rose, C.A.F.D.: A performance isolation analysis of disk-intensive workloads on container-based clouds. In: Euromicro International Conference on Parallel (2015)
21. Li, Y., Zhou, M., You, C., Yang, G., Mei, H.: Enabling on demand deployment of middleware services in componentized middleware. In: Grunske, L., Reussner, R., Plasil, F. (eds.) CBSE 2010. LNCS, vol. 6092, pp. 113–129. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13238-4_7
22. Zhang, J., Sivasubramaniam, A., Riska, A., Qian, W., Riedel, E.: An interposed 2-level i/o scheduling framework for performance virtualization. In: ACM Sigmetrics International Conference on Measurement & Modeling of Computer Systems (2005)
23. Zhu, T., Tumanov, A., Kozuch, M.A., Harchol-Balter, M., Ganger, G.R.: Prioritymeister: Tail latency QOS for shared networked storage. In: ACM Symposium on Cloud Computing (2014)