AOMO: An AI-aided Optimizer for Microservices Orchestration

Xue Leng[†‡], Tzung-Han Juang[‡], Yan Chen[‡], and Han Liu[‡]

[†] Zhejiang University, China [‡] Northwestern University, USA

CCS CONCEPTS

• **Networks** → **Cloud computing**; • **Applied computing** → *Service-oriented architectures.*

KEYWORDS

Microservices, Orchestration Optimization

1 INTRODUCTION

Microservices as a new service-oriented architecture attracts more attention from industry and academia, and there are 91% of companies are using or have plans to use microservices [4]. Compared to the traditional monolithic service architecture, microservice architecture decouples applications based on the functionality of modules. Benefiting from the loosely coupled architecture, applications can achieve multi-language development, independent deployment, fast iteration, and flexible management. The large companies usually have a huge number of microservices to support business operations, e.g., Twitter has $O(10^3)$ microservices and $O(10^5)$ microservice instances [5]. Hence, managing these large number of microservices at the lowest labor costs and equipment costs is not only a goal but also a challenge for Microservices Management Service Providers (MMSPs).

Compared to traditional cloud services, microservices are more dynamic and have a larger scale. Hence, a microservices orchestrator should have the following properties:

- **Scalability.** Orchestrating a huge number of microservices is a challenge. The scheduling plan should be given in constant time to achieve rapid deployment.
- **Agility.** Microservices are lightweight and dynamic. Hence, the orchestrator needs to scale microservices agilely based on the workload and resources usage of microservices and servers.
- **High Resource Utilization.** Fully utilizing the resources (e.g. CPU and memory) and reducing the number of servers can significantly reduce the costs of MMSPs. However, improving the resource utilization without introducing degradation in the quality of service is still a challenge.

There are many state-of-the-art works which focus on only one or two properties mentioned above. As illustrated in Table 1, model-based work ([3, 6]) build models to get an optimal solution, which can achieve high resource utilization but cannot be applied to deploying large-scale microservices due to high computational complexity. And the heuristic-based work [2] is a trade-off solution, which can work in large-scale scenarios but only obtain a suboptimal solution and have limited resource utilization. The best way to manage the huge number of microservices is to adopt management tools like Kubernetes, Spring Cloud, and Docker Swarm. Nevertheless, their scheduling mechanisms have some drawbacks. For instances, the current scheduling and scaling mechanisms only refer to the instantaneous resource usage of servers instead of the historical resource usage of microservices and servers, which may lead to the underutilization of resources when the workload is low, and the risk of resource contention when most microservices are under high workload. Besides, existing scaling mechanisms are not agile enough and not fully automatic when scaling microservices, which makes the scaling lag and needs human intervention.

Table 1: AOMO vs. Existing Work

Solutions	Scalability	Agility	High Utilization
model-based work [3, 6]	x	x	✓
heuristic-based work [2]	✓	✓	x
AOMO	✓	✓	✓

To address the downsides mentioned above and minimize the total costs of MMSPs, in this paper, we propose **AOMO**, an **A**I-aided **O**ptimizer for **M**icroservices **O**rchestration. Under the premise of guaranteeing the QoS of Service Consumers, AOMO is designed to deploy microservices with the minimal number of nodes[1], as well as automatically scale pods[2] based on the fluctuation of microservices' workload. By collecting and analyzing the historical resource usage of nodes and pods, AOMO can predict the resource usage of each node for current and after a period of time. Based on the prediction, AOMO leverages the pairwise ranking model to calculate a near optimal microservices deployment plan in constant time, which can work well for large-scale microservices deployment. Moreover, AOMO leverages the prediction of workload and resource usage of running microservices to complete pre-scaling before the system reaches the critical state. This automatic scaling can improve the resource utilization of the cluster while ensuring the quality of service. Since AOMO improves the resource utilization, reduces the number of nodes running in the cluster, and realizes automatic pre-scaling, the equipment costs and labor costs of MMSPs can be significantly reduced.

2 DESIGN

To minimize the costs of MMSPs, AOMO automatically deploys and pre-scales microservices with the minimal number of nodes. As shown in Fig. 1, the *Data Collector* collects the system metrics

SIGCOMM Posters and Demos '19, August 19–23, 2019, Beijing, China

ACM ISBN 978-1-4503-6886-5/19/08…$15.00
https://doi.org/10.1145/3342280.3342287

[1] *Node* is the machine to run microservices, which may be a physical machine or virtual machine.

[2] *Pod* is the smallest creation and deployment unit consisting of one or more containers that are tightly coupled and share resources.

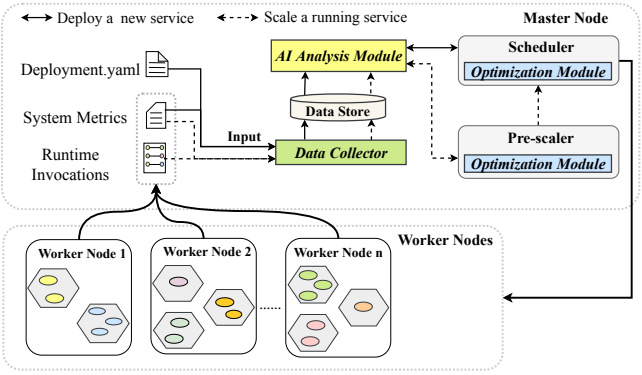

Figure 1: System framework.

(e.g., the CPU and Memory usage of nodes and pods, the resource capacity of each node, and the pod allocation of each node) periodically and store them to the *Data Store*. When deploying a new service, *Scheduler* assigns nodes considering the requirement of each undeployed pod and the capacity of nodes in the cluster. After services are deployed successfully, *Pre-scaler* monitors the workload and resource utilization of each microservice and scales the microservice in advance once a prediction that the critical state is about to be reached is detected.

2.1 Intelligent Scheduler.

Scheduler is responsible for selecting a node for newly created pods to run on. Optimizing the deployment of microservices can significantly reduce the migration costs. We adopt pairwise ranking to allocate pods to the node with the highest priority. As represented in Eq. (1), for each node pair (N_j, N_k), *Scheduler* takes the resource requirement of pod (R_{pod}) and the current resource capacity of nodes (C_{N_j}, C_{N_k}) as input to calculate the ranking between two nodes. After all comparisons are completed, the node N_i with the highest priority will be selected to run the pod.

$$P(N_j \geq N_k | R_{pod}, C_{N_j}, C_{N_k}) = \frac{1}{1 + e^{-f_\omega(R_{pod}, C_{N_j}, C_{N_k})}} \quad (1)$$

The current scheduler follows the one-by-one manner to schedule pods, whose allocation plan is affected by the arrival order of pods and is not optimal. Different from the current scheduler, AOMO *Scheduler* follows the p-batch manner, which each time takes p unscheduled pods as input to calculate the allocation plan with the ranking model. Since the pods can be deployed in parallel, the scheduling time of microservices will be reduced significantly.

2.2 Pre-scaler.

Scaling microservices based on the demands and workload can improve the resource utilization and reduce costs. However, existing scaling mechanisms only consider instantaneous resource usage and cannot realize fully automatic scaling, which either has the hysteresis of scaling or needs human intervention. Take the Fig. 2 as an example, developers define the desired state of microservice s, and s reaches the critical state at t_1, then the microservice management tool like Kubernetes detects this issue at t_s and start to scale up pods for microservice s, thus the resource utilization

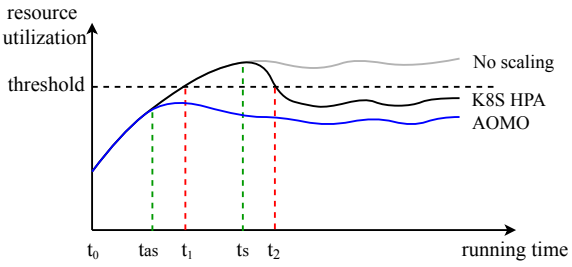

Figure 2: Pre-scaling Timeline.

of s falls back under the threshold at t_2. Although this mechanism can achieve auto-scaling for microservices, it inevitably makes the system be in a critical state for a while.

To address the shortcoming of auto-scaler, we design a proactive prediction-based *Pre-scaler*, which can automatically scale the microservices based on resource usage predictions in advance. AOMO collects the CPU and memory usage of pods and nodes continuously, then *Pre-scaler* takes these data as input to train the Bidirectional Long Short-Term Memory (BI-LSTM) model, and a prediction will be generated that microservice s will reach the critical state at t_1, the scaling should be completed before that time. Finally, the *Pre-scaler* scales the microservice s at t_{as} in advance before reaching the critical state. This mechanism can ensure the system always in the ideal state and automatically scale microservices based on the prediction of the workload and resource usage.

3 PRELIMINARY RESULTS

We implement a prototype of AOMO and train the prediction model with the dataset *Alibaba Cluster Data V2017* [1], which provides the resource (i.e., CPU and memory) capacity of each server, the requested resource of each container, and the resource usage of containers and servers which are collected from about 1300 machines for 24 hours. We also evaluate the performance of the *Data Collector*. Collecting system metrics for 1 hour, it only consumes 4.89 millicpu and 8.18 MB memory and occupies 336 KB storage space.

ACKNOWLEDGEMENTS

This work is supported in part by the National Key R&D Program of China (2017YFB0801703), and in part by the Key Research and Development Program of Zhejiang Province (2018C01088).

REFERENCES

[1] Alibaba. 2017. Alibaba Cluster Data v2017. (2017). http://bit.ly/alidata2018 Accessed on 2019-06-06.
[2] Carlos Guerrero and Isaac Lera, *et al.* 2018. Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications. *The Journal of Supercomputing* (2018), 1–28.
[3] Gueyoung Jung and Matti A Hiltunen, *et al.* 2010. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *2010 30th ICDCS*. IEEE, 62–73.
[4] LightStep. 2018. Global Microservices Trends: A Survey of Development Professionals. (2018). http://bit.ly/trendsofms Accessed on 2019-04-29.
[5] Benedict Michael and Charanya Vinu. 2017. How we built a metering and charge-back system to incentivize higher resource utilization of twitter infrastructure. (2017). http://bit.ly/twitteryoutubems Accessed on 2019-04-29.
[6] Adalberto R Sampaio and Julia Rubin, *et al.* 2019. Improving microservice-based applications with runtime placement adaptation. *Journal of Internet Services and Applications* 10, 1 (2019), 4.