CO-TRIS: Container Orchestration – Transforming container using Resource Inspection System

Joo-Young Roh
Computer & Information Secuirty
Sejong University
Seoul, South Korea
hpjoo718@gmail.com

Sang-Hoon Choi
Computer & Infromation Secuirty
Sejong University
Seoul, South Korea
csh0052@gmail.com

Ki-Woong Park*
Computer & Information Secuirty
Sejong University
Seoul, South Korea
woongbak@sejong.ac.kr
*Corresponding author

***Abstract*— Traditional approaches to container orchestration, such as distributed threshold-based policies, often result in isolated or underutilized resources. This can negatively affect application availability and performance. This paper presents a new orchestration framework that solves the cloud-computing resource-allocation problem, focusing on containerized environments. This framework focuses on the runtime phase of the container lifecycle. It uses visualized resource information to optimize resource reallocation in cloud services, and introduces a systematic orchestration approach that enables efficient resource utilization and real-time allocation. Its goal is to improve the smooth operation and quality of service (QoS) of containers and microservices in a cloud environment. Our method improved the experimental allocation from 24 container blocks to 67. The results improved linearly until 30k epochs and then plateaued.**

Keywords— Cloud Systems, System Design, CaaS(Container as a service), Orchestration, Reinforcement Learning, Deep Learning

I. INTRODUCTION

The shift from cloud systems to cloud-native systems is in full swing, with an emphasis on infrastructure resilience, flexibility, and efficiency [1]. Cloud-native services and microservices create an environment for modular, flexible development and rapid iteration; however, they present complementary operational aspects and can cause new system problems in terms of resource operation and management [2, 3]. This has increased the importance of maintenance and resource monitoring.

To ensure the smooth operation and quality of service (QoS) of containers and microservices, infrastructure development and operations (DevOps) managers have developed various approaches to optimize resource usage and systematically build and operate application runtime environments [4, 5]. One such approach is resource orchestration [6, 7]. Container-orchestration tools automate the provisioning, management, communication, and deployment of containers, helping to manage the entire lifecycle of distributed applications, such as containers and microservices. Kubernetes is a typical orchestration tool [8, 9]. Container-orchestration tools use distributed threshold-based policies that employ a system metric to set thresholds for container resources. However, such a policy can over-allocate resources to running applications, resulting in allocated resources being unused and isolated. This can lead to reduced application availability. Ongoing research addresses this problem [10, 11, 12, 13].

We proposed and designed an orchestration framework that can efficiently reallocate containers to run more containers with limited physical resources. The framework monitors the resource usage in a running container environment, visualizes it in block form, determines the optimized container placement using machine learning, and migrates the container.

II. RELATED STUDIES

A. Research on resource gathering

The first is the computation and time-delay issues caused by resource collection. Because our main idea is to orchestrate the use of blocked container resources, it is important to efficiently collect running container resources. In this section, we introduce related research on container resource collection.

cAdvisor (Container Advisor): cAdvisor is a component designed to collect system matrices from containers [14]. It runs independently of other processes to collect resources and interacts directly with the runtime API and OS to collect matrices. It provides the ability to aggregate, store, and visualize the collected resource data. Kubernetes, the most representative container-orchestration tool, leverages cAdvisor for resource collection.

eBPF (extended Berkeley Packet Filter): Originally used at the network layer to capture packets and analyze traffic, eBPF [15] can now perform tasks, such as tracking, monitoring, security, and performance analysis, in the kernel [16, 17, 18, 19].

2831-7203/23/$31.00 ©2023 IEEE
DOI 10.1109/AIKE59827.2023.00027

eBPF is being leveraged to build custom tools to track containers and collect matrices for resource utilization, network communication, and application behavior.

As a representative example of using eBPF to collect resources, Red Hat introduced a process for combining Performance Co-Pilot (PCP) and bpftrace to perform continuous resource collection [20]. Moreover, ViperProbe [21] proposed an eBPF-based scalable dynamic and adaptive microservice resource-collection framework.

B. Research on resource orchestration

Container orchestration refers to the automated configuration, management, and control of containerized applications[22, 23].

Kubernetes, which can manage these matters, serves as an intermediary to continuously support and manage the smooth operation of instance and container states. YAML or JSON files representing the desired state of the application are deployed and applied to working nodes. These files contain detailed information about the Docker image to run in the container, required resource assignments, ports and services, etc. The YAML or JSON file created to initiate the orchestration process is published on the Kubernetes API server, where the master node efficiently distributes workloads across the cluster to satisfy the desired state [23]. Once configured, Kubernetes distributes the workload throughout the cluster and successfully establishes containerized nodes.

C. Limitations of previous studies

When monitoring resources using existing container-orchestration tools, the traditional approach retrieves the entire resource metric from the API server. This process can create unnecessary overhead [24]. This study leverages eBPF technology to reduce the overhead associated with existing resource collection. It filters the resources to only collect those specific to the proposed framework, to reduce unnecessary data retrieval and optimize the performance and resource utilization.

Orchestration typically allocates resources using fixed thresholds. However, such static orchestration cannot accommodate workloads where resources are allocated, but idle or underutilized. These "standby resources" do not contribute to workloads and can reduce the application performance [10, 12].

To mitigate the limitations of existing orchestration tools, this paper proposes a reinforcement-learning-based orchestration framework that uses adaptive thresholds and predictive analytics to gather specific resources, optimize resource allocation, and increase resource utilization.

III. Framework Structure

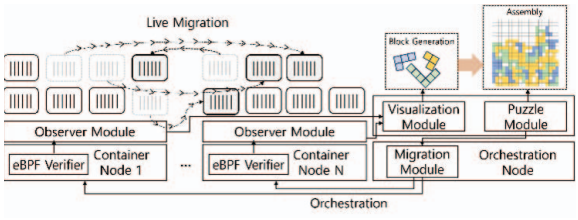

Fig. 1. CO-TRIS framework structure

To address the problem of resource overallocation, owing to the structural problems in orchestration tools that were discussed as limitations in existing research, we propose the following orchestration-methodology framework named CO-TRIS. The proposed framework consists of four major modules: 'Observer,' 'Visualization,' 'Puzzle,' and 'Migration', as shown in Figure 1.

A. Observer module

The 'Observer' module uses eBPF technology to filter and collect data related to the specific resources in each container for a desired resource. The 'Observer' module then preprocesses the collected matrices to prepare them for visualization in subsequent modules. In this study, we focused on the orchestration methodology; we plan to develop an eBPF resource-collection module in the future.

B. Visualization module

The 'Visualization' module receives the data preprocessed by the 'Observer' module and represents the data as they change in real time. The 'Visualization' module allows system administrators and operators to quickly understand resource-usage distributions and patterns for effective container placement and resource management.

C. Puzzle module

The 'Puzzle' module combines blocked resource data using reinforcement-learning techniques and leverages learned knowledge and algorithms to find the most efficient way to reallocate resources, while maximizing utilization within the given constraints. It allocates resources effectively to minimize the "standby resources" in a dynamic environment, where the resource availability and demand change over time.

D. Migration module

The 'Migration' module applies the settings and configuration determined in the 'Puzzle' module to migrate a container or resource from its current location to a new, optimized location.

IV. Experiments

The experiments in this study were intended to validate the CO-TRIS orchestration methodology. The concept of block-based container relocation was validated through the following experiments. We imaged the collected resources as a combination of 2D blocks and then experimented with placing the blocks generated by each container in a single space. The following assumptions were made.

First, each block was confined to encompass no more than 20 cells, ensuring that the overall dimensional constraint of the image remained under 20 cells. Each cell represented one block unit. In this study, the space in which blocks could be placed was limited to a 20 × 20 grid. This restriction is because of the limitations of our GPU. We conducted experiments using image shapes to represent the resource usage of three types of containers and utilized a reinforcement-learning model for placement. In our resource-visualization approach, a diverse set of 24 distinct block configurations was examined.

A. Orchestration Experiment

To validate our reinforcement learning-based approach for resource orchestration, we conducted the following experiments. Initially, we tested three containers, each consisting of three basic blocks, as shown in the top section of Figure 2. In these blocks, the Y-axis represents memory and the X-axis represents the CPU.

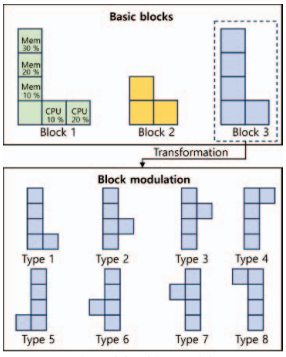

Fig. 2. Container blocking with resource

The baselines for the axes are CPU and memory, which are the representative resources that create the containers and can directly affect performance. Each cell in this block corresponds to a percentage value. For example, Block 1 represents 20% CPU usage and 30% memory usage. An additional cell is added for each 10% increase in usage. The identified blocks can also be reorganized in eight different ways. We utilized a total of 24 container blocks. Eight variations are shown at the bottom of Figure 2, all of which were derived from Block 3.

To demonstrate the learning progress of these container blocks, we developed an orchestration scheme similar to that of Tetris. The scheme was trained using the deep-queue network (DQN) [25] algorithm, which combines the principles of deep and reinforcement learning. The parameters used in the learning process are listed in Table I.

TABLE I. Reinforcement-learning parameters

Learning Rate	1e-3
Batch Size	512
Gamma	0.99
Epochs	85,000

B. Orchestration-experiment results

Figure 3A illustrates the initial reinforcement-learning state, in which 24 container blocks were allocated at the onset of the learning process. Conversely, Fig. 3B depicts the postlearning state. Substantial free space was eliminated, as depicted in the figure. This resulted in the allocation of 67 containers.

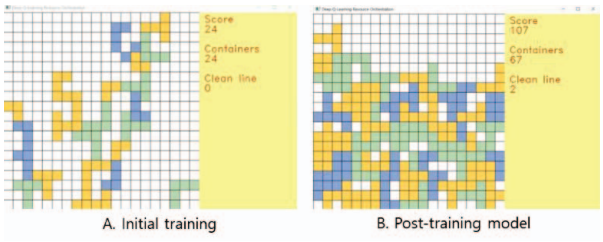

Fig. 3. Container orchestration with reinforcement learning

Figure 4 tracks the evolution of the score over the epochs, indicating that after training 85k instances, an average score of 83 was attained. The scoring was based on the following criteria: 1 point was assumed for each block of containers placed. The rule of adding 10 points to a clean line was applied. The score displayed linear growth from the commencement of learning until 30k, after which it plateaued with no marked improvement. This stagnation is attributed to the geometric constraints of the blocks.

Given that only three block shapes were employed, there was an inherent structural limitation in forming a clear line using only these three configurations. Addressing these challenges requires the incorporation of blocks that extend beyond two-dimensional configurations.

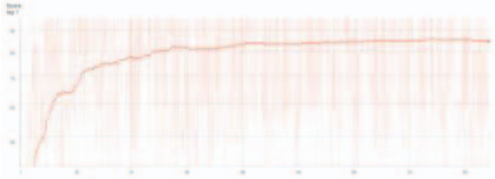

Fig. 4. Container orchestration with reinforcement learning

V. Conclusion

Container-orchestration tools control container resources with fixed threshold policies. This can lead to the over-allocation of resources and reduce application safety and performance because of unused "standby resources." To address these issues, this study collected container resources driven by Kubernetes in a container-as-a-service (CaaS) environment, visualized them in blocks, and reassembled them for space allocation.

A limitation of the block-relocation experiments in this study was that the resources collected in the block-visualization phase were represented in one dimension. Moreover, the limitations of the axes allowed the resources to be defined only as blocks.

A more diverse container block shape should be visualized in an actual application-centric container environment. However, the limitations of 24 block shapes and the short learning time in the experiment made it difficult to confirm the applicability of the framework to a diverse environment.

In future research, we will develop a resource-collection tool using eBPF, mentioned in the 'Observer' module,' and compare the performance and measure the overhead between the existing resource-collection process (Docker API, cgroup) and the developed tool. In addition, we will solve the problem of the geometry of resource blocks in this study by enabling the visualization of N-dimensional puzzle geometry, based on the collected resources, and increase the number of learning cases by applying the framework to various environments. Furthermore, we plan to extend the application of this framework to cloud-native services and microservices.

Acknowledgment

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) ,(Project No. RS-2022-00165794, Development of a Multi-Faceted Collection-Analysis-Response Platform for Proactive Response to Ransomware Incidents, 30%), (Project No.2022-0-00701, 10%, Project No.RS-2023-00228996, 10%), the ICT R&D Program of MSIT/IITP, (Project No. 2021-0-01816, A Research on Core Technology of Autonomous Twins for Metaverse, South Korea, 10%), and a National Research Foundation of Korea (NRF), South Korea grant funded by the Korean government (Project No. RS-2023-00208460, 40%)

References

[1] Balalaie, Armin, Abbas Heydarnoori, and Pooyan Jamshidi. "Microservices architecture enables devops: Migration to a cloud-native architecture." IEEE Software vol. 33, issue 3, pp. 42-52 May 2016.

[2] Gan, Yu, et al. "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems." Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 3-18, April 2019.

[3] Zhou, Hao, et al. "Overload control for scaling wechat microservices." Proceedings of the ACM Symposium on Cloud Computing. pp. 149-161, October 2018.

[4] McDaniel, Sean, Stephen Herbein, and Michela Taufer. "A two-tiered approach to I/O quality of service in docker containers." 2015 IEEE International Conference on Cluster Computing. 29 October 2015..

[5] Xu, Cong, Karthick Rajamani, and Wesley Felter. "Nbwguard: Realizing network qos for kubernetes." Proceedings of the 19th international middleware conference industry. pp. 32-38, 10 December 2018.

[6] Ranjan, Rajiv, et al. "Cloud resource orchestration programming: overview, issues, and directions." IEEE Internet Computing, vol. 19, issue 5, pp. 46-56, 2015.

[7] Daradkeh, Tariq, and Anjali Agarwal. "Adaptive Micro-service based Cloud Monitoring and Resource Orchestration." 2022 13th International Conference on Information and Communication Systems (ICICS), pp. 127-132, 04 July 2022.

[8] kubernetes, 2014, https://kubernetes.io/.

[9] Bernstein, David. "Containers and cloud: From lxc to docker to kubernetes." IEEE cloud computing, vol. 1, issue 3, pp. 81-84, September 2014.

[10] Bauer, André, et al. "Chamulteon: Coordinated auto-scaling of micro-services." 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp.2015-2025, 31 October 2019.

[11] Beltrán, Marta. "Automatic provisioning of multi-tier applications in cloud computing environments." The Journal of Supercomputing, vol. 71, pp.2221-2250, 21 January 2015.

[12] Jayesh Vartak. "Eliminate Kubernetes node scaling lag with pod priority and over-provisioning" https://aws.amazon.com/ko/blogs/containers/eliminate-kubernetes-node-scaling-lag-with-pod-priority-and-over-provisioning/, 13 January 2023.

[13] Rossi, Fabiana, Valeria Cardellini, and Francesco Lo Presti. "Hierarchical scaling of microservices in kubernetes." 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), pp.28-37, 15 September 2020.

[14] Chang, Chia-Chen, et al. "A kubernetes-based monitoring platform for dynamic cloud resource provisioning." GLOBECOM 2017-2017 IEEE Global Communications Conference. December 2017.

[15] eBPF(Extended Berkeley Packet Filter), 2011, https://ebpf.io.

[16] Deri, Luca, et al. "Combining System Visibility and Security Using eBPF." ITASEC, vol. 2315, 2019.

[17] Liu, Chang, et al. "A protocol-independent container network observability analysis system based on eBPF." 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), pp. 697-702, December 2020.

[18] Vieira, Marcos AM, et al. "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications." ACM Computing Surveys (CSUR), vol. 53, issue 1, pp. 1-36, 06 February 2020.

[19] Cassagnes, Cyril, et al. "The rise of eBPF for non-intrusive performance monitoring." NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1-7, April 2020.

[20] Karl Abbott. "Visualizing System Performance with RHEL 8 Part 3: Kernel Metric Graphing with Performance Co-Pilot, Grafana, and Bpftrace." https://www.redhat.com/en/blog/visualizing-system-performance-rhel-8-part-3-kernel-metric-graphing-performance-co-pilot-grafana-and-bpftrace, 3 February 2021.

[21] Levin, Joshua, and Theophilus A. Benson. "ViperProbe: Rethinking microservice observability with eBPF." 2020 IEEE 9th International Conference on Cloud Networking (CloudNet), pp. 1-8, November 2020.

[22] Zhong, Zhiheng, et al. "Machine learning-based orchestration of containers: A taxonomy and future directions." ACM Computing Surveys (CSUR), vol. 54, issue 10s, pp.1-35, 13 September 2022.

[23] Nguyen, Thanh-Tung, et al. "Horizontal pod autoscaling in Kubernetes for elastic container orchestration." Sensors vol. 20 issue 16, pp.1-18, 17 August 2020.

[24] Medel, Víctor, et al. "Characterising resource management performance in Kubernetes." Computers & Electrical Engineering, vol. 68, pp.286-297, May 2018.

[25] Osband, Ian, et al. "Deep exploration via bootstrapped DQN." Advances in neural information processing systems 29, pp.1-9, 2016.