International Journal of Advanced Smart Convergence Vol.12 No.4 126-133(2023)
http://dx.doi.org/10.7236/IJASC.2023.12.4.126

IJASC 23-4-15

An Engine for DRA in Container Orchestration Using Machine Learning

Gun-Woo Kim*, Seo-Yeon Gu**, Seok-Jae Moon***, Byung-Joon Park****

**Master, Department of Computer Science, Kwangwoon University, Korea*
***Master, Department of Computer Science, Kwangwoon University, Korea*
****Professor, Department of Artificial Intelligence Institute of Information Technology, KwangWoon University, Korea*
*****Professor, Department of Computer Science, Kwangwoon University, Korea.*
E-mail: {kgw0528, gsy0207, msj8086, bjpark}@kw.ac.kr

Abstract

Recent advancements in cloud service virtualization technologies have witnessed a shift from a Virtual Machine-centric approach to a container-centric paradigm, offering advantages such as faster deployment and enhanced portability. Container orchestration has emerged as a key technology for efficient management and scheduling of these containers. However, with the increasing complexity and diversity of heterogeneous workloads and service types, resource scheduling has become a challenging task. Various research endeavors are underway to address the challenges posed by diverse workloads and services. Yet, a systematic approach to container orchestration for effective cloud management has not been clearly defined. This paper proposes the DRA-Engine (Dynamic Resource Allocation Engine) for resource scheduling in container orchestration. The proposed engine comprises the Request Load Procedure, Required Resource Measurement Procedure, and Resource Provision Decision Procedure. Through these components, the DRA-Engine dynamically allocates resources according to the application's requirements, presenting a solution to the challenges of resource scheduling in container orchestration.

Keywords: *Container Orchestration, Cloud Computing, Machine learning, Resource Provisioning, Integrated Environmental Management*

1. INTRODUCTION

Recently, cloud service providers have transitioned from a focus on Virtual Machine technology to container-centric cloud computing technology, offering advantages such as rapid deployment, high portability, and minimal resource usage [1]. Microservices-based architecture involves individual applications and services for each virtual container, utilizing physical resources and enabling mutual communication [2]. Container orchestration, a key technology for efficient management of these containers, has been proposed. However, scheduling becomes increasingly challenging over time due to diverse workloads, including heterogeneous applications and services [3]. Consequently, issues such as resource imbalance, excessive or

Manuscript Received: october. 12, 2023 / Revised: october. 18, 2023 / Accepted:october. 23, 2023
Corresponding Author: msj8086@kw.ac.kr
Tel: +82 -10-7753-8086, Fax: +82-050-4366-4238
Author's affiliation Master, Department of Computer Science, Kwangwoon University, Korea

wasteful usage of specific container resources, and degradation of service quality arise [4]. To address these challenges, various studies are underway focusing on maintaining high provisioning quality, resource improvement, efficient distribution, energy consumption reduction, and meeting user Quality of Service (QoS) requirements [5]. However, a systematic approach and management method for addressing these issues have not been proposed thus far [5]. In this paper, we propose the DRA-Engine (Dynamic Resource Allocation Engine) for resource scheduling optimization. It consists of three procedures: Request Load Procedure, Required Resource Measurement Procedure, and Resource Provision Decision Procedure. In the Request Load Procedure, workloads and resource quantities requested by users are loaded from storage. The Required Resource Measurement Procedure utilizes a Fuzzy Rule-Based Neural Network to measure the required resources for the requested workloads. Based on the measured values, the Resource Provision Decision Procedure calculates QoS suitability and determines appropriate resource provisioning using rule-based methods. This dynamic resource measurement process enables container resource scheduling tailored to the application's requirements. The paper is organized as follows: In Chapter 2, we review related studies, followed by a description of the proposed container orchestration engine configuration and FLOW in Chapter 3. Chapter 4 details the performance analysis of the proposed engine, and finally, Chapter 5 concludes the paper.

3. RELATED WORK

In the cloud environment, various machine learning-based models are comprehensively researched for predicting resource usage of heterogeneous workloads [1]. The utilization of machine learning and artificial intelligence technologies in cloud computing enables improved results for a wide range of complex issues, such as energy optimization and workflow management. Furthermore, it can contribute to enhancing various aspects of existing cloud computing systems, including dynamic load management, task scheduling, energy optimization, real-time migration, and cloud security. To achieve this, various approaches for integrating cloud and machine learning have been proposed [8].

The increase in cloud resource volume and its heterogeneous nature have made resource scheduling a critical issue in cloud computing. Inefficient scheduling techniques can lead to service performance degradation due to excessive resource usage or imbalance. Additionally, issues such as waste and scarcity of cloud resources may arise. The fundamental idea of scheduling is to fairly distribute diverse and complex tasks among cloud resources, allowing scheduling algorithms to avoid imbalance issues. Scheduling algorithms must optimize key performance metrics such as response time, processing time, reliability, availability, energy consumption, cost, and resource utilization. To achieve this, many recent scheduling algorithms based on heuristics, metaheuristics, and hybrids have been proposed [3].

3. PROPOSED SYSTEM

3.1 Proposed System Overall Components

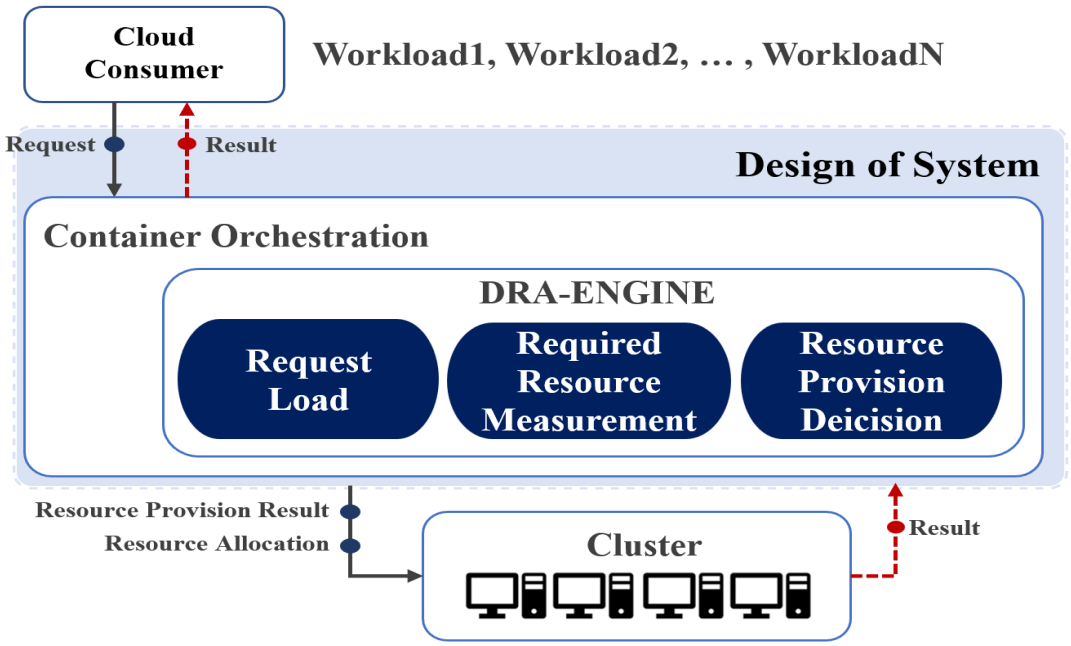

Figure 1. Proposed System Overall Structure

In this section, we delve into the configuration of the DRA-Engine in container orchestration. Figure 1 illustrates this structure. The DRA-Engine is comprised of three components: the Request Load Procedure, the Required Resource Measurement Procedure, and the Resource Provision Decision Procedure.

- **Request Load Procedure (RLP).** In this procedure, workloads and resource amounts requested by users are loaded, and the validity of this information is verified. Users request resource scheduling in container orchestration for workloads that include heterogeneous applications and services. Subsequently, one of the orchestration functionalities, Node Monitor, conveys the user's request to the RLP. The RLP loads information about the requested workloads and resource amounts from the data storage and checks the validity of the information.

- **Required Resource Measurement Procedure (RRMP**). This procedure measures the required resource levels for each workload using the loaded information. It is crucial to efficiently process resource consumption information from heterogeneous services and applications. To achieve this, a neural network model [6] can be employed to measure the required resource levels. The neural network model, pre-trained to measure how much resources a process in the workload requires, is utilized. This approach determines the optimal amount of resources tailored to each workload. Additionally, various techniques such as Principal Component Analysis and Gaussian Mixture Model may be employed through the neural network model to effectively obtain results.

- **Resource Provision Decision Procedure (RPDP).** This procedure determines the optimal resource provisioning for each node and provides it to the task scheduler. Various decision factors (QoS, SLA, Node Stat, ETC) are considered to comply with Quality of Service (QoS) and Service Level Agreement (SLA) agreements for each user. Based on the analyzed required resource amounts of workloads from the previous

RRMP and various decision factors, it calculates the QoS fitness (F) and the fitness without considering decision factors (Fnon) for each workload. Each workload will have different results based on their QoS fitness values. If the QoS fitness function F is less than the non-considered Fnon value, the process goes through the entire cycle again through renegotiation with the user. If the conditions are met, resource provisioning is determined based on rule-based decision making. The determined provisioning values undergo node state checks for appropriate resource distribution.

3.2 DRA-Engine Sequence Diagram

Figure 2 represents the data flow of the proposed DRA-Engine in a sequence diagram. This diagram illustrates the overall flow of the DRA-Engine and is repeated each time there is a new request from the user.

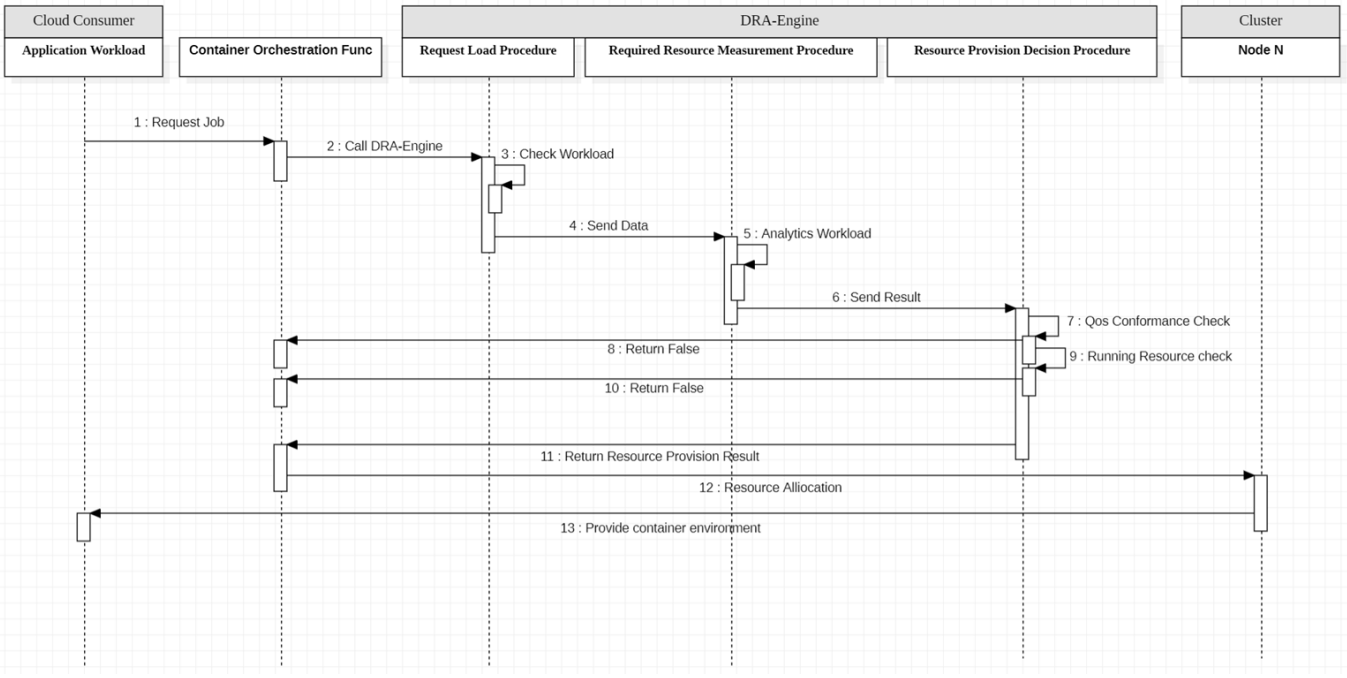

Figure 2. DRA-Engine Sequence Diagram

1. Request Workload: Container orchestration receives a new workload request from the user, including applications and services.

2. Call DRA-Engine: Container orchestration requests resource provisioning from the DRA-Engine.

3. Check Workload: The Request Load Procedure loads and validates the received workload.

4 ~ 5: Once the Request Load Procedure is complete, the Required Resource Measurement Procedure utilizes the Neural Network to measure the required resource amounts for the workload.

6. Send Result: The measured values are sent to the next step to determine resource provisioning.

7 ~ 8. QoS Conformance Check: Rule-based QoS fitness values are calculated using required resource amounts and various decision factors (QoS, SLA, Node Stat, ETC). If the QoS fitness value does not meet a specific threshold, the process is halted and returned.

9 ~ 10. Running Resource Check: Determines whether resources can be allocated on the current node while

meeting SLA agreements. If SLA agreements are not met, it may return to the workload analysis phase, or the administrator can modify resource allocation mid-process.

11 ~ 13: Once resource provisioning is determined, it is deployed to each node through container orchestration functionalities. Users can then utilize the deployed resources.

3.3 DRA-Engine Algorithm

Algorithm 1 lists the sequential execution of each procedure in the DRA-Engine. The Request Load Procedure is abbreviated as RLP, the Required Resource Measurement Procedure as RRMP, and the Resource Provision Decision Procedure as RPDP. D represents the user's request, J represents the workload, and Q is a variable used for processing QoS values. The DRA-Engine is implemented in Python.

Algorithm 1. Sequence Algorithm

```
Input : Workload, QoS, SLA,ETC
Output : ResourceProvisioningResult
BEGIN:
# Load from storage via user request D and verify that it is valid data.
# If the data is valid, J,Q will return their respective values; otherwise, None will be returned.
# A value of None means that no value was received due to an error.
# Returns J for workload and Q for QoS.
    J, Q = DRA.RLP.load_check(D)
# Return an error if Q has the value None.
    if Q == None || J == None:
        return check_input_data

# A Neural Network Model is loaded to measure the amount of resources needed.
    DRA.RRMP.Call()
# If the amount of resources required is measured, it is returned in measures_result.
    measures_result = DRA.RRMP.Measurement(J)

# Compute the value of the fitness function F given the workload J and the QoS Q value.
# Also compute the value without considering the QoS value (Fnon).
     F, Fnon = DRA.RPDP.QoSCalculate(measures_result,Q)
# If the value of F is less than the value of Fnon, renegotiate with the user and start over.
     if Fnon > F:
         return need_renegotiate

# Use the measured value to determine the resource provisioning value.
     ResourceProvisionResult = DRA.RPDP.RunningResourceCheck(measures_result);
# Return the value to container orchestration for resource allocation.
     return ResourceProvisionResult
END
```

4. EXPERIMENTS AND RESULTS

In this chapter, two experiments were conducted to analyze the performance of the proposed DRA-Engine. The first experiment compared the proposed system with a heuristic system based on particle swarm optimization (PSO) [7], a common approach, using the same workload. The analysis focused on differences in overall response time, including request time, processing time, and completion time. The second experiment measured the overall response time of concurrent workload requests for different node counts in the proposed system, analyzing the workload concurrency capacity for each node count. The experimental environment is detailed in Table 1. The user-requested workload involved analyzing regional carbon emission costs for scientific analysis purposes, provided by the Environmental Big Data Platform, with an average processing time of 300 seconds.

Table 1. Experiment Specification

Item	Details
Processor	Intel(R) Core(TM) i9-12900 @ 3.20GHz - 16
Memory	64 GB
Storage	2 TB HDD
Network	10 Gbit/s network card
OS	Ubuntu 20.04

The first experiment involved specifying the number of nodes within the agent as 4 and using the same set of 12 workloads. The total response time was measured for each workload.

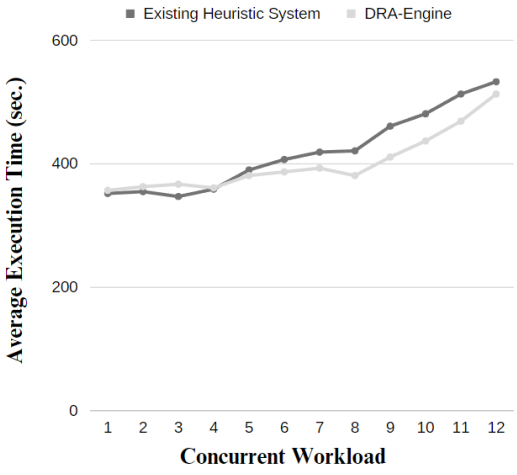

Figure 3. Comparison of average overall response times for workflows

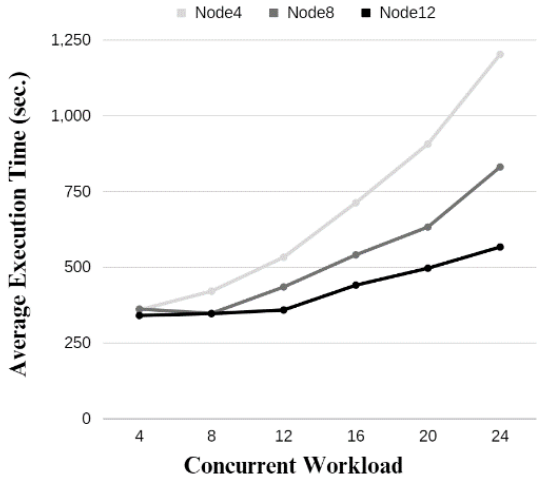

Figure 4. Response time depending on number of nodes

Figure 3 is a line chart comparing the average response times between the existing heuristic scheduling policy system and the proposed DRA-Engine. Twelve workloads were used with 4 nodes to record and analyze concurrent job requests (J = 1,2, ..., 12). When there were fewer concurrent job requests (J<5), both approaches showed similar performance. However, with more job requests (4<J<13), it can be observed that the job request

time of the proposed DRA-Engine is faster in terms of total response time than the system with the existing policy. This indicates that in a small workload environment, the proposed system is more efficient in resource scheduling than the existing system in a specific range (4<J<13). However, when resources are exhausted on all nodes, the average response time increased significantly for both the existing system and the DRA-Engine approach.

Figure 4 represents the second experiment, showing a table measuring response times according to the number of nodes. Environments with 4, 8, and 12 nodes were established, measuring concurrent job response times for the same workload with J (J = 4, 8, 12, 16, 20, 24). It can be observed that the increase in response time becomes more significant when J=20 or higher with 8 nodes and when J=8 with 4 nodes. This indicates the DRA-Engine's ability to handle concurrent workloads depending on the number of nodes. If the number of concurrent job requests reaches a certain threshold for each node count, there is a significant increase in the total response time. This is because, similar to the experiment in Figure 3, resources were exhausted on all nodes, but there were still unprocessed workloads causing delays. Additionally, delays are expected due to communication between multiple nodes and a single central server. Through the results of these two experiments, it can be confirmed that the DRA-Engine is more suitable for small workload environments compared to the existing system.

5. CONCLUSION

In this paper, we propose the DRA-Engine in the context of container orchestration to address the challenges of resource scheduling arising from increasingly complex services and heterogeneous workloads. The proposed engine consists of three procedures: the Request Load Procedure, Required Resource Measurement Procedure, and Resource Provision Decision Procedure, designed to dynamically allocate resources according to the application's demands. Through comparative experiments using overall response time, we observed that in resource-constrained environments, the proposed engine outperformed conventional container scheduling techniques in certain segments. However, as resource constraints intensified with a higher workload, delays occurred, resulting in performance similar to existing methods. This suggests that in small-scale workload environments, more efficient resource scheduling is achievable compared to conventional heuristic systems. Future research should focus on optimizing algorithms for each procedure and exploring additional research on heterogeneous services and lightweight mechanisms.

Acknowledge

This work is financially supported by Korea Ministry of Environment(MOE) Graduate School specialized in Integrated Pollution Prevention and Control Project.

References

[1] A. M. Potdar, N. D G, S. Kengond, and M. M. Mulla, “Performance Evaluation of Docker Container and Virtual Machine,” Procedia Computer Science, vol. 171, pp. 1419–1428, Jun 2020, DOI: https://doi.org/10.1016/j.procs.2020.04.152

[2] C. Carrión, “Kubernetes Scheduling: Taxonomy, ongoing issues and challenges,” ACM Computing Surveys, vol.55, pp. 1-37, July 2023, DOI: https://doi.org/10.1145/3539606

[3] Mohit Kumar, S.C. Sharma, Anubhav Goel, S.P. Singh, "A comprehensive survey for scheduling techniques in cloud computing", Journal of Network and Computer Application, vol. 143, pp. 1-33, June 2019, DOI: https://doi.org/10.1016/j.jnca.2019.06.006

[4] D. Saxena, J. Kumar, Ashutosh Kumar Singh, and S. Schmid, "Performance Analysis of Machine Learning Centered Workload Prediction Models for Cloud," IEEE Transcations on Parallel and Distributed Systems, vol. 34, no. 4, pp. 1313–1330, April 2023, DOI: https://doi.org/10.1109/tpds.2023.3240567

[5] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions," ACM Computing Surveys, vol. 54, no.217, pp. 1-35, Sep 2022, DOI: https://doi.org/10.1145/3510415

[6] P. V. de Campos Souza, "Fuzzy neural networks and neuro-fuzzy networks: A review the main techniques and applications used in the literature," Applied Soft Computing, vol. 92, p. 106275, July 2020, DOI: https://doi.org/10.1016/j.asoc.2020.106275

[7] Arabinda Pradhan, Sukant Kishoro Bisoy, Amardeep Das, "A survey on PSO based meta-heuristic scheduling mechanism in cloud computing environment," Journal of King Saud University - Computer and Information Sciences, vol. 34, pp. 4888-4901, Jan 2022 DOI: https://doi.org/10.1016/j.jksuci.2021.01.003

[8] Yogesh Kumar, Surabhi Kaul, Yu-Chen Hu, "Machine learning for energy-resource allocation, workflow scheduling and live migration in cloud computing: State-of-the-art survey," Sustainable Computing: Informatics and Systems, vol. 36, p. 100780, Jan 2022 DOI: https://doi.org/10.1016/j.suscom.2022.100780