International Journal of Advanced Culture Technology Vol.12 No.3 471-479 (2024)
DOI https://doi.org/10.17703/IJACT.2024.12.3.471

IJACT 24-9-56

Research on Hot-Threshold based dynamic resource management in the cloud

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

**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: {kgwo0528, msj8086, bjpark}@kw.ac.kr

Abstract

Recent advancements in cloud computing have significantly increased its importance across various sectors. As sensors, devices, and customer demands have become more diverse, workloads have become increasingly variable and difficult to predict. Cloud providers, connected to multiple physical servers to support a range of applications, often over-provision resources to handle peak workloads. This approach results in inconsistent services, imbalanced energy usage, waste, and potential violations of service level agreements. In this paper, we propose a novel engine equipped with a scheduler based on the Hot-Threshold concept, aimed at optimizing resource usage and improving energy efficiency in cloud environments. We developed this engine to employ both proactive and reactive methods. The proactive method leverages workload estimate-based provisioning, while the reactive Hot-Cold Scheduler consists of a Predictor, Solver, and Processor, which together suggest an intelligent migration flow. We demonstrate that our approach effectively addresses existing challenges in terms of cost and energy consumption. By intelligently managing resources based on past user statistics, we provide significant improvements in both energy efficiency and service consistency.

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

1. INTRODUCTION

Cloud computing is becoming increasingly significant with the advancement of virtualization technologies. Notably, the shift from virtual machines (VMs) to container-based systems marks a critical change in the cloud computing environment, enhancing scalability, flexibility, efficiency, and resource utilization [1]. Resource management in cloud computing remains a critical issue and is directly dependent on application workloads. Applications are often connected to specific physical servers in traditional cloud computing environments like data centers and are over-provisioned to handle issues related to maximum workloads [2]. Servers or hosts in data centers are assigned workloads with various types of workloads across multiple VMs. The variability and unpredictability of workloads can lead to overutilization or underutilization of servers, causing imbalances in

Manuscript received: July 27, 2024 / revised: August 25, 2024 / accepted: September 10, 2024
Corresponding Author: msj8086@kw.ac.kr
Tel: ***-****-**** Fax: +82-050-4366-4238
Professor, Department of Artificial Intelligence Institute of Information Technology

resource utilization for VMs on specific hosting servers. This results in inconsistent service quality (QoS), imbalanced energy use, and violations of service level agreements (SLA) [3]. Additionally, imbalanced workloads can decrease productivity in data centers, leading to increased energy consumption, which correlates with operational costs and financial losses. Excessive energy consumption also directly impacts the carbon footprint [4]. Various approaches have been proposed to address workloads with nonlinear and variable behaviors, including methods based on historical workload data to generate insights through trend learning and homeostasis-based prediction methods that provide future workload insights by subtracting current workloads from previous ones [5]. To address these issues, intelligent resource management strategies are needed to meet QoS requirements and enhance data center efficiency. This paper proposes an engine centered around the Hot-Threshold Scheduler based on the Hot-Threshold concept to improve resource utilization efficiency in VMs. The Hot-Cold based Scheduler system uses both proactive and reactive approaches. The proactive approach involves allocating physical resources based on workload estimates to enhance resource utilization and energy efficiency, utilizing the concept of resource provisioning [6]. The reactive approach calculates each virtual machine's resource usage using machine learning in the Feature Impact Module and General Estimate Module, based on past usage statistics of each virtual machine, using the Hot-Cold concept to decide whether to reallocate resources or migrate based on specific thresholds. This intelligent resource management minimizes SLA violations while improving energy use and operational cost efficiency. The paper is organized as follows: Section 2 reviews related research, Section 3 describes the proposed Hot-Threshold based system architecture and flow, Section 4 discusses the performance analysis of the proposed Hot-Threshold Scheduler, and Section 5 concludes the paper.

2. RELATED WORK

Scheduling is one of the key issues in cloud computing. Research on cloud computing scheduling has explored various models, including elasticity, usage-based pricing models, demand-based services, and multi-user environments. Traditional scheduling consists of two phases: resource allocation and job scheduling. Resources are selected based on quality of service parameters, resource allocation involves selecting a suitable VM instance for the job, and finally, the VM instance is allocated to an available host or physical machine. Resource allocation is related to load balancing, and job scheduling concerns the optimal job order according to scheduling objectives. Traditional algorithms such as First-Come, First-Served (FCFS) and Shortest Job First (SJF) have been proposed, but their performance has declined due to the diversification of sensors and equipment and the complexity of workloads. Scheduling in cloud computing is an NP-hard problem, and meta-heuristic algorithms like Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) have helped in finding approximate optimal solutions. However, with the increase in problem space, the limitation of getting stuck in local optima has been observed. Research on maintaining load balance by finding the optimal solution is presented in [9-11]. In this paper, we use the Fast Up and Slow Down (FUSD) algorithm to apply suitable solutions for the system, employing thresholds for Hot, Warm, Cold, Sleep states is shown in Table 1 [12].

Table 1. Parameters in simulation

symbol	meaning	value
h	hot threshold	0.9
w	Warm threshold	0.65
c	Cold threshold	0.4
s	Sleep limit	0.05

3. PROPOSED SYSTEM

Figure 1 illustrates the entire system structure based on the DRA-Engine [6], which includes the Hot-Threshold Scheduler. The DRA-Engine periodically collects resource usage statistics for each VM on each node. It operates in two modes: a proactive approach and a post-management approach.

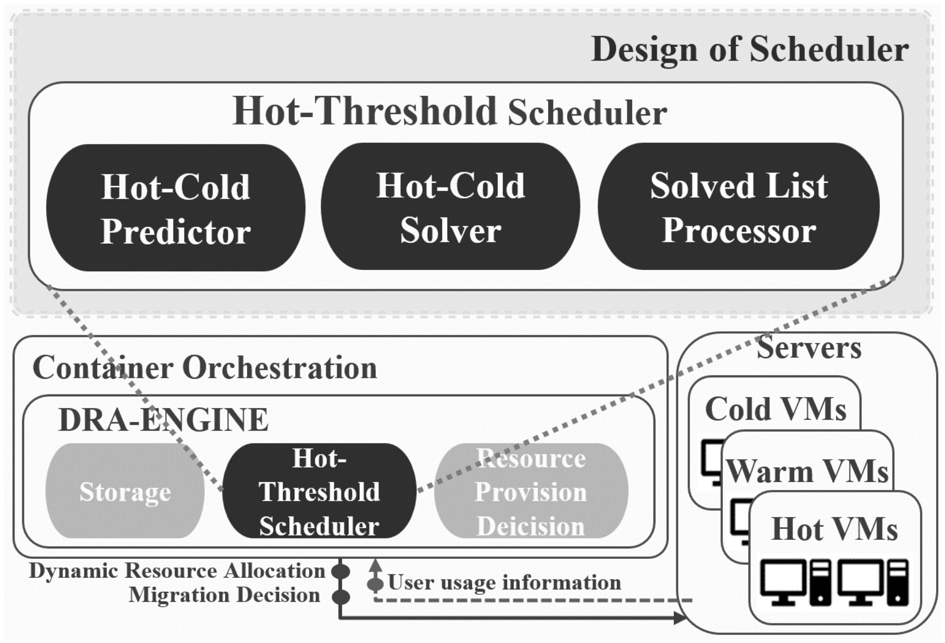

Figure 1. Proposed system overall structure

The proactive approach enhances resource utilization and energy efficiency by creating and deploying containers with suitable resources based on estimates after analyzing users' SLAs and workloads through the Dynamic Resource Provision Module of the existing DRA-Engine [6]. The post-management approach uses the Hot-Threshold concept to minimize SLA violations, such as availability and response time, while improving energy use and operational cost efficiency through intelligent resource management. Based on this concept, the Hot-Threshold Scheduler is composed of multiple components. It calculates the resource efficiency of each virtual machine through machine learning in the Feature Impact Module and General Estimate Module, using past statistical data of each virtual machine. Resources are then reallocated or migrated based on specific threshold values. The Hot-Threshold Scheduler is called periodically and performs actions appropriate for the future state of VMs. The overall structure of the Hot-Threshold Scheduler is presented in Section 3.1 through Figure 2. This module consists of the Hot-Cold Predictor, Hot-Cold Solver, and Solved List Processor.

3.1 Proposed Scheduler Overall Components

- **Hot-Cold Predictor.** In this procedure, the future resource demands of VMs and the future load of PMs are measured based on past usage statistics data. The usage statistics data of virtual machines belonging to each physical server are periodically batch-saved in Storage. When the Hot-Cold Predictor procedure is called, it first retrieves the relevant statistics data from Storage. Based on the retrieved statistics data, the future resource demands of each virtual machine for each server are measured through the General Estimation Module and Feature Impact Module. To address anomalous patterns and complex workloads, a stack ensemble method is used [8]. Different prediction models are employed to measure information from various aspects, achieving higher prediction accuracy than a single model. The measurement results are calculated as the sum of the results from the Estimate Module and the Feature Impact Module, normalized to ensure consistent outcomes before calculation. The Estimate Module measures future resource usage based on supervised learning methods, including users' past usage patterns and workload information for each container. Past usage patterns are represented as vectors indicating time-based access, while workload information includes categorical types of processes and data on how much resource a process consumes over time periods. The Feature Impact Module is used to emphasize the importance of specific features (frequency of users' specific process usage), ranking users' importance. This enables accurate hot-cold data prediction. The sum of the results from the Feature Impact Module and the General Estimation Module helps to understand future resource usage patterns and establish an efficient resource allocation plan.

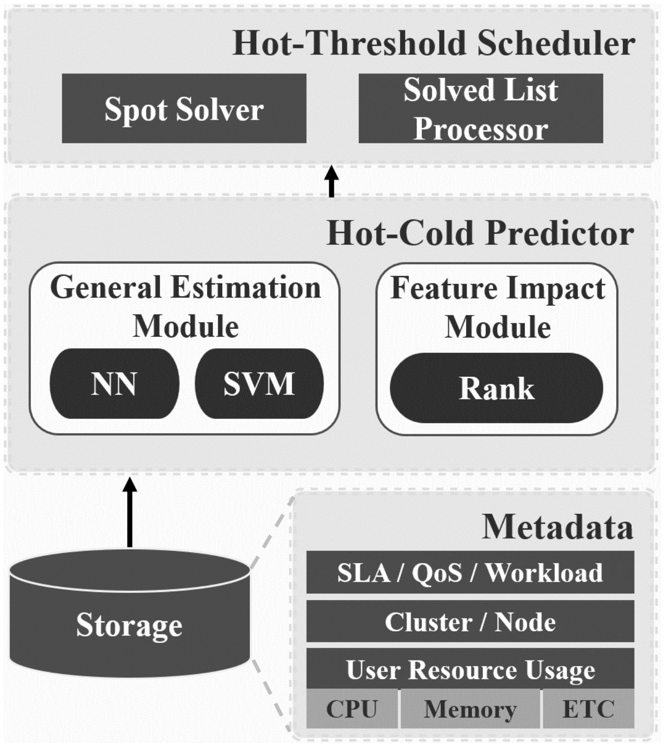

Figure 2. Proposed hot-threshold scheduler overall sequence

- **Hot-Cold Solver.** This module aggregates the future resource demands for each container and determines the status (Hot, Cold, Sleep, Warm) of each physical server (PM) based on that. The determination of each physical server's status is based on comparing the future resource usage values measured by the Predictor with the threshold values listed in Table 1. For each VM, this module uses the future resource usage values to calculate and determine the status of each PM, which is then forwarded to the Solved List Processor.

- **Solved List Processor.** When the Solved List Processor procedure is called, servers identified as Hot Spots act first, followed by Cold Spots, Sleep Spots, and Warm Spots, in that order. Servers classified as Hot Spots, due to their high resource utilization, will have some of their VMs migrated to other physical servers to reduce load. Physical servers identified as Cold Spots, showing inefficiently low utilization, will either have their running VMs migrated to physical servers identified as Warm Spots for energy savings or be switched to a standby mode. This strategy allows for operational cost and resource savings. Sleep Spots refer to servers in standby mode, while Warm Spots indicate physical servers with moderate utilization.

3.2 System Sequence Diagram

Figure 3 depicts the operational flow of the proposed Hot-Threshold Scheduler as a sequence diagram. It not only illustrates the functioning of the Hot-Threshold Scheduler but also represents the overall operational flow of the system. The diagram specifically shows the post-management approach, excluding the proactive approach..

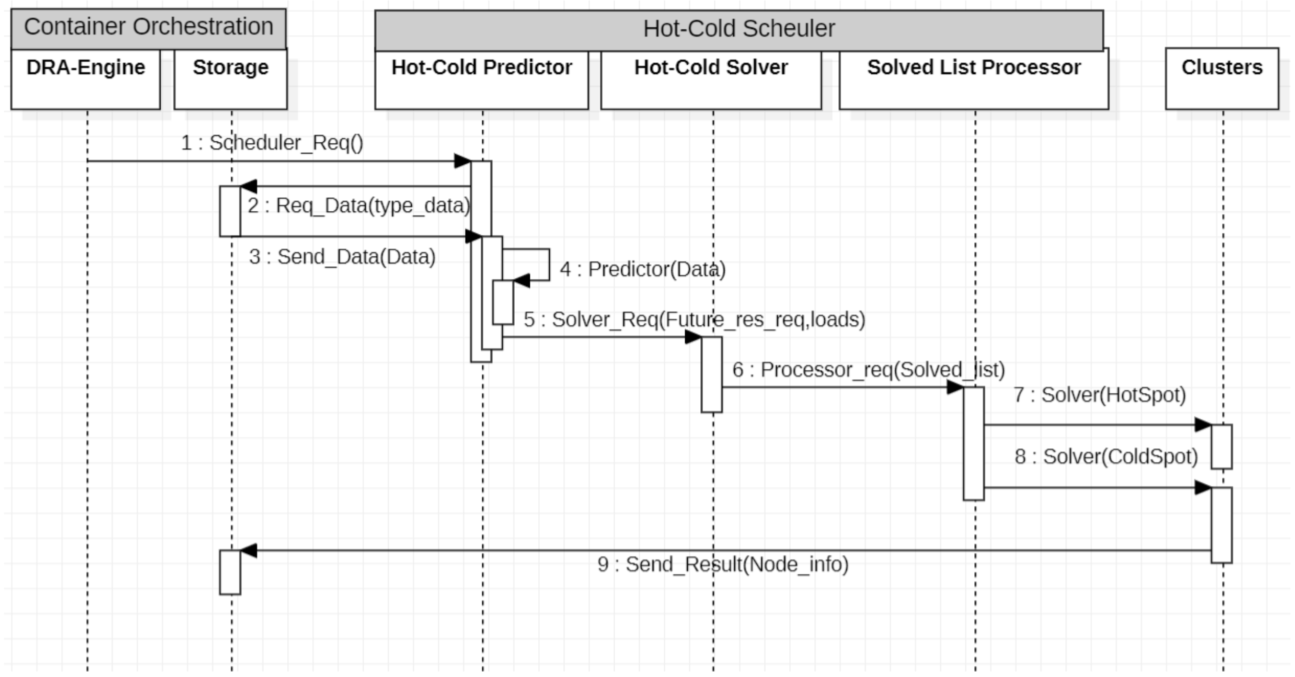

Figure 3. System sequence diagram

1. Scheduler_Req(): Every 10 minutes, the DRA-Engine periodically invokes the Hot-Threshold Scheduler.
2. Req_Data(type_data): The Hot-Cold Predictor requests historical usage statistics data from the storage by sending a 'type_data' value to measure the future resource demands of VMs..
3. Send_data(Data): Upon being called, the storage sends the requested data based on the received argument values.
4. Predictor(Data): Using the historical usage statistics data received from storage for each virtual machine, it calculates the future resource demands of VMs and the future load of PMs. The calculation process utilizes the Estimate Module and the Feature Impact Module.
5. Solver_Request(Future_res_req_loads): The future resource demand values measured by the Hot-Cold Predictor are compiled and sent to the Hot-Cold Solver to classify each physical server as Hot, Warm,

Cold, or Sleep.

6. Processor_req(Solved_list): The Hot-Cold Solver sends the Solved_list values, which have completed the classification of each server as Hot, Warm, Cold, or Sleep, to the Solved List Processor. The called Solved List Processor then invokes each action in the order of HotSpot, ColdSpot, etc. As a result, each physical server performs actions appropriate to its state.
7. Solver(HotSpot): The Solver function receives a list of servers identified as HotSpot. To reduce the resource utilization of physical servers classified as HotSpot, some virtual machines are migrated to suitable physical servers.
8. Solver(ColdSpot): The Solver function receives a list of servers identified as ColdSpot. To save energy and cost resources on physical servers classified as ColdSpot, some virtual machines are migrated to suitable physical servers, and if necessary, switched to a standby mode.
9. Send_Result(Node_info): After the scheduling process concludes, each server sends its current status to the DRA-Engine.

3.3 Hot-Threshold Scheduler Algorithm

Algorithm 1 and 2 respectively outline the proactive and post-management approaches through formulas, detailing the execution of each procedure. The Hot-Threshold Scheduler has been implemented in JAVA for experimentation with CloudSim [7].

Sequence Algorithm 1 Hot-Threshold Scheduler

```
1:  INPUT : UserUsageStatisticsData, SLA, QoS, loads
2:  OUTPUT : SolvedListProcessorResult
3:  BEGIN:
4:  # BEGIN BY DRA-ENGINE SIGNAL (Receive a signal every 10 minutes)
5:  # type_data is a value for requesting the user's past usage statistics data from storage.
6:  # The user's past usage statistics data is data collected and periodically monitored by the
    engine.
7:  Data = DRA.Storage.Request_Data(type_data);
8:  # The user's past usage statistics data is data collected and periodically monitored by the
    engine
9:  # Predictor operates with the General Estimate Module and Feature Impact Module.
10: # The return value of Predictor contains the future resource usage for each container.
11: Future_Resource_Usage = DRA.HT_Scheduler.Predictor(Data);
12: Solved_list = DRA.HT_Scheduler.Solver_Request(Future_Resource_Usage, loads);
13: # Processor_Request is classified into HotSpot and ColdSpot using Solved_list.
14: # Afterwards, the Solver function is used to execute HotSpot and then ColdSpot in that order
    to perform resource allocation and migration as needed.
15: # Solver.Call(HotSpot), Solver.Call(ColdSpot)
16: SolvedListProcessorResult = DRA. HT_Scheduler.Processor_Request(Solved_list)
17: # SolvedListProcessorResult has the result value of the Solver() function operation.
18: return SolvedListProcessorResult
19: END
```

4. EXPERIMENTS AND RESULTS

In this Section, the performance of the proposed Hot-Threshold Scheduler was analyzed using simulations with CloudSim [7]. The simulation structure consisted of 3 host servers, with each host deploying 4 virtual machines. Each virtual machine and host was set up in identical environments as shown in Table 2 The scheduling methods were implemented through functions written in Java code. User statistics data, including daily and hourly patterns of CPU usage rates, memory usage rates, and more, were randomly generated for the simulation purposes.

Table 2. 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 22.04

In this experiment, the efficiency of overload mitigation, energy, and resource savings within a small-scale environment was evaluated. The experiment simulated a flash crowd event by generating random data and inducing load by increasing the CPU and memory usage of virtual machines (VMs) [13]. The Hot-Threshold Scheduler proposed in this study detects VMs with increased load and activates standby host servers to appropriately distribute the load.

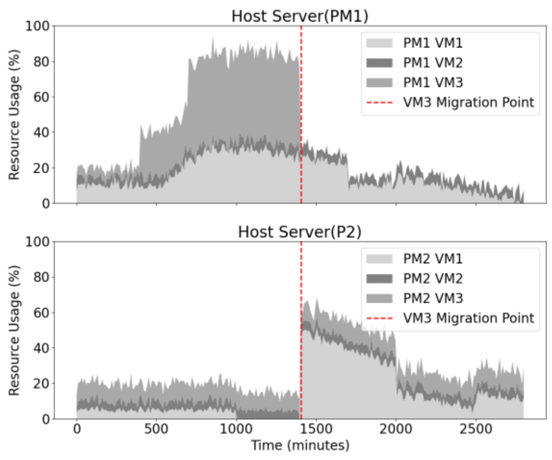

Figure 4. Resource balance in host

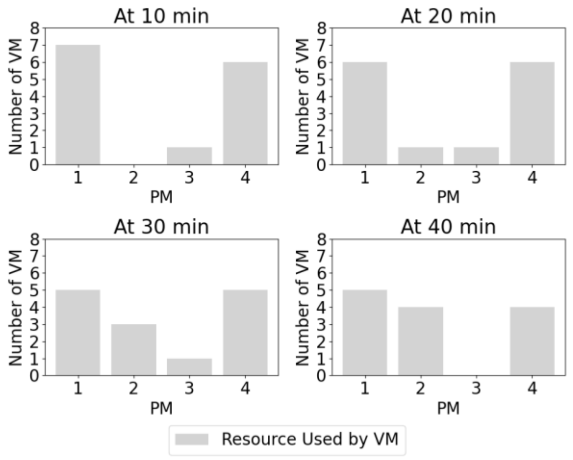

Figure 5. VM resource balance over time

The results of this experiment are detailed in Figure 4, which shows how two physical machines and three virtual machines operate based on CPU and memory usage. Every 10 minutes, the scheduler operates, and the graph illustrates the process of transferring the load from VM3 of PM1 to VM1 of PM2. This demonstrates that when a continuous workload of the same size and resource consumption is input into a specific VM, the distribution of resources between PMs is adjusted to manage CPU and memory loads evenly. In further experiments, the scale of the environment was expanded to include a total of 4 host servers, with each host having

8 VMs, making a total of 32 VMs, to conduct the experiment. By gradually applying load to PM2 and PM4, it was observed how the load was distributed to other PMs over time. Since the scheduler operates on a 10-minute cycle, measurements were taken four times based on this interval. As can be seen in the results from Figure 5, it was observed that distributing the load over time to other PMs increased the efficiency of resource and energy usage. This demonstrates that the Hot-Threshold Scheduler maintains system stability through appropriate migration to prevent overload situations, even with continuous load, and puts unnecessary parts into standby mode to enhance energy and resource efficiency. However, delays can occur when there is a continuous workload and usage requests to the extent that the system cannot cope.

5. CONCLUSION

In this paper, we proposed the Hot-Threshold Scheduler within the DRA-Engine [6] environment for intelligent resource scheduling in cloud environments. We utilized a post-management approach with a Hot-Cold classification method. For intelligent resource management, we used past usage statistics data for each virtual machine to calculate future resource usage for each physical server through the General Estimate Module and the Feature Impact Module. This allowed us to determine whether a server is Hot, Cold, Warm, or Sleep and to perform actions appropriate to its status. Through our experiments, we confirmed that it is possible to appropriately balance the load by distributing it to other PM servers when a specific VM experiences a load in a small-scale environment. However, since we only used the stacking ensemble technique in analyzing past user usage statistics, further research is needed to explore better machine learning methods. In future work, we plan to compare various machine learning models, examine traffic distribution in real cloud environments, and consider both Quality of Experience (QoE) and Quality of Service (QoS).

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. Joy, "Performance comparison between Linux containers and virtual machines," 2015 International Conference on Advances in Computer Engineering and Applications, IEEE, pp. 342-346, March 2015, doi: 10.1109/ICACEA.2015.7164727.

[2] M. Xu, W. Tian, and R. Buyya, "A survey on load balancing algorithms for virtual machines placement in cloud computing," Concurrency and Computation: Practice and Experience, p. e4123, October 2017, doi: doi.org/10.1002/cpe.4123.

[3] A. K. Singh and J. Kumar, "Secure and energy aware load balancing framework for cloud data centre networks," Electronics Letters, pp. 342-346, March 2015, doi: doi.org/10.1049/el.2019.0022.

[4] L. A. Barroso, U. Hölzle, and P. Ranganathan, "The Datacenter as a Computer," Synthesis Lectures on Computer Architecture, Springer, pp. 1-189, August 2013, doi: 10.1007/978-3-031-01761-2.

[5] J. Kumar and A. K. Singh, "Workload prediction in cloud using artificial neural network and adaptive differential evolution," Future Generation Computer Systems, Elsevier, pp. 41-52, October 2018, doi: doi.org/10.1016/j.future.2017.10.047.

[6] G.-W. Kim, S.-Y. Gu, S.-J. Moon, and B.-J. Park, "An Engine for DRA in Container Orchestration Using Machine Learning," International Journal of Advanced Smart Convergence, vol. 12, no. 4, pp. 126-133, December 2023, doi: doi.org/10.7236/IJASC.2023.12.4.126.

[7] R. N. Calheiros, et al., "Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services," March 2009, doi: doi.org/10.48550/arXiv.0903.2525.

[8] A. Dixit, Ensemble Machine Learning: A Beginner's Guide That Combines Powerful Machine Learning Algorithms to Build Optimized Models, 2017.

[9] M. H. Shirvani, "A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems," Engineering Applications of Artificial Intelligence, vol. 90, 2020, Art. no. 103501, doi: 10.1016/j.engappai.2020.103501.

[10] N. Mansouri, B. M. H. Zade, and M. M. Javidi, "Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory," Computers and Industrial Engineering, vol. 130, pp. 597-633, 2019, doi: 10.1016/j.cie.2019.03.006.

[11] H. M. Alkhashai and F. A. Omara, "BF-PSO-TS: Hybrid heuristic algorithms for optimizing task scheduling on cloud computing environment," International Journal of Advanced Computer Science and Applications (IJACSA), vol. 7, no. 6, pp. 207-212, 2016, doi: 10.14569/ijacsa.2016.070626.

[12] Z. Xiao, W. Song, and Q. Chen, "Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1107-1117, June 2013, doi: 10.1109/TPDS.2012.283.

[13] Wiktionary, "Flash crowd," *Wiktionary*, accessed September 2024. Available: en.wiktionary.org/wiki/flashcrowd.