RESEARCH ARTICLE **SPECIAL ISSUE**

RSP Science Hub

International Research Journal on Advanced Science Hub
2582-4376
Vol. 05, Issue 05S May
www.rspsciencehub.com

Check for updates

http://dx.doi.org/10.47392/irjash.2023.S058

International Conference on intelligent COMPUting TEchnologies and Research (i-COMPUTER) 2023

A Bio inspired Approach for Load Balancing in Container as a Service Cloud Computing Model

Kodanda Dhar Naik [1], Rashmi Ranjan Sahoo [2], Sanjay Kumar Kuanar [1]

[1]Department of Computer Science and Engineering, Gandhi Institute of Engineering and Technology University, Gunupur, Odisha, India.

[2]Center of Excellence on Cyber Security and Cloud Computing, Department of Computer Science and Engineering, Parala Maharaja Engineering College (Govt. of Odisha), Berhampur, Odisha, India.

Emails: kodandadhar.naik@giet.edu kd_naik@yahoo.co.in, rashmiranjan.cse@pmec.ac.in, sanjay.kuanar@giet.edu

Article History

Received: 28 February 2023
Accepted: 18 March 2023

***Keywords**:*

Cloud computing;
Containerization;
CaaS;
Honey bee mating algorithm;
Load Balancing;
Task scheduling

Abstract

In recent years, container-based virtualization has gained popularity due to its ease of deployment and agility in cloud resource provisioning. The traditional virtual machine (VM) is based on modern innovation, has superseded technology in cloud computing which is known as containerization technology, and it is superior in terms of overall performance, reliability and efficiency. Containerized clouds deliver superior performance because they make the most of the resources available at the host level and make use of a load-balancing strategy. In order to accomplish this goal, the focus of this article is on equitably dividing of the workload across all of the available servers. In this research, we proposed a Honeybee Mating Algorithm (HBMA) to combat the issue of load balancing in the container-based cloud environment by considering the deadline of tasks. We compared our findings to those of the Grey Wolf Optimization (GWO), Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) Algorithms. We assessed the performance of the proposed methods by considering the impact of parameters such as load variation and makespan. According to the findings of our proposed method, almost the tasks were completed within the deadline, and the HBMA performed significantly better than any of the other strategies in terms of load variance and makespan.

1. Introduction

Cloud computing is a model for enabling multiple users to pool and share an organization's physical and logical computing resources over the Internet (Dimitri). Since it operates on a pay-as-you-go basis, it has attracted a large customer base. Pay-as-you-go is a business strategy in which clients pay in advance for the resources or services they will utilize (Dimitri). Users can adjust the number of available resources to meet their changing business needs while service costs are reduced. Obtaining any service from the cloud requires the user to sign up in the service provider portal and submit a web request (Niyogi, Chronopoulos, and Zomaya). It is the responsibility of the cloud service provider to manage the available computing resources and respond to requests from various clients. The cloud service provider implements numerous scheduling strategies for effective resource management (Gawali and Shinde). Cloud computing's allocation and scheduling of resources has a profound impact on the efficacy of all its oper-

ations. As a result, cloud-based job scheduling, and resource allocation have attracted the attention of numerous academics (Gawali and Shinde).

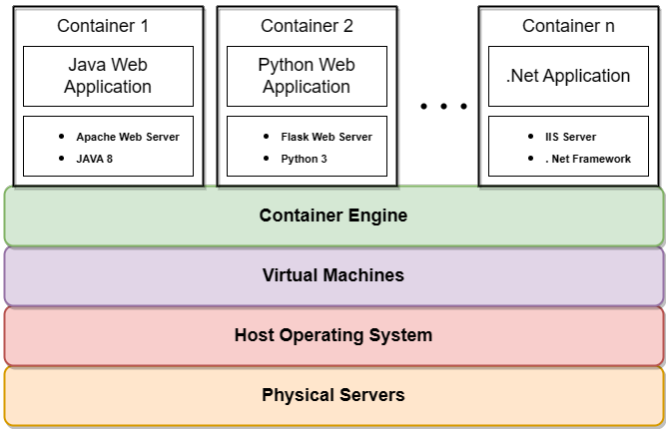

FIGURE 1. CaaS Model

Containers as a Service (CaaS) is a new cloud service model developed by various cloud service providers in addition to the more traditional Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Services (SaaS) models. When it comes to cloud computing, a container is a virtualization method at the OS-level (Maenhaut et al.). The image of a container is an independent, lightweight, executable package of software that contains the application code, runtime, settings system tools and libraries (Patra et al.). It has everything that's required to run the program. The OS-level virtualization is built on containers, which are the essential building blocks (Maenhaut et al.). It eliminates the requirement for a hypervisor or other monitoring middleware by creating sandboxed virtual environments. With the help of an OS container or an application container, it is possible to implement virtualization at the OS-level (Maenhaut et al.). In this work, we focus on the operating system container. When it comes to the packaging of various resources like languages, databases, and libraries, OS containers are the way to go. When the program is to be packaged as a component, an application container is an ideal choice. Figure 1 depicts containerization.

The CaaS paradigm executes tasks within a container scheduled in virtual machines (VMs), with VMs deployed on a physical system. Figure 1 depicts the architecture of a CaaS paradigm, which combines system and OS-level virtualization. Under a CaaS approach, customers can leverage OS-level virtualization to operate, manage, resize, upload, and organize containers. It is very portable because once a container for an application is created, it has all the components required to run the application. Security is one of the primary reasons for utilising the CaaS model. Compared to containers, VMs are more isolated. So, they will be more isolated and secure if we group them and deploy them on multiple VMs rather than operating all of the containers on a single server. It enables the user to use a different private or public cloud environment to run that application. This makes it simple for the user to transit between cloud service providers. In the cloud computing services hierarchy, CaaS is in the middle, between IaaS and PaaS. CaaS is a subset IaaS. Google Kubernetes and Docker Swarm are two major CaaS orchestration solutions.

The primary goal of task scheduling is to make optimal use of available resources and to arrange incoming requests in such a way that they may all be carried out by the given deadline. With cloud computing, several clients can share the same computational resource and run multiple tasks simultaneously. As a result, smaller tasks may miss their deadline and take a long time to complete if the system does not use an appropriate scheduling mechanism. The following are the primary contributions:

- Honey bee mating based evolutionary algorithm is proposed to handle the issue of load balancing in the container orchestration based cloud platform.
- The load variance and make span attributes are considered to compare the efficacy of the proposed model with other evolutionary based containerized cloud model.

The remaining sections of the paper are laid out as follows: Section 2 presents the related research and development efforts made in cloud-based load balancing and resource allocation. Different CaaS cloud architecture components are illustrated in Section 3. In Section 4, we detail the suggested system model and algorithms. The findings and discussions from the proposed work are presented in Section 5. In Section 6, we conclude with the proposed work.

2. Related Work

Several meta-heuristics can be used to identify optimal solutions to numerous optimization problems. It has become increasingly common to utilize meta-heuristic algorithms to find optimal solu-

tions to problems (Sanshi, D., and Vatambeti). Meta-heuristic techniques are attractive for complex problem-solving because they can quickly and effectively deliver good solutions even for enormously huge problems.

In order to improve the efficiency of cloud systems, it is important to schedule tasks and distribute workloads among all processing nodes. In (T. Wang et al.), the author presented a double-fitness adaptive method for task scheduling and load-balancing genetic algorithm (JLGA). In (B. Wang and Li), a Multi- Population Genetic Algorithm (MPGA) is used to schedule jobs in a cloud environment while maintaining fair load distribution. In (Babbar et al.), genetic load-balancing technique is proposed for multimedia applications, which reduces user wait times by spreading the workload over multiple servers.

In (Pradhan and Bisoy), A. Pradhan et.al. suggested a modified PSO method to decrease makespan and maximize resource consumption and load balance. Load balancing and energy efficiency were proposed by FIMPSO (Elhoseny et al.). This algorithm reduces search space and improve responsiveness, throughput, makespan, execution time, and resource consumption. In (Alguliyev, Imamverdiyev, and Abdullayeva), authors presented a mechanism to efficiently move VM overload duties to equivalent cloud VMs. The optimization model minimizes task execution and transfer time. They aim to reduce transfer and execution time. (Alguliyev, Imamverdiyev, and Abdullayeva) proposes a PSO-based static task scheduling technique for cloud-based independent and non-preemptive workloads.

In (Hussein, Mousa, and Alqarni), Hussein, M.K. et.al. proposed an ant colony optimization (ACO) based hybrid model for container placement in cloud environments to reduce the makespan of container scheduling and load balancing across the physical server. A GWO- based technique was suggested by authors in (Sefati, Mousavinasab, and Farkhady) to the load of server balanced by considering reliability and capacity of resources. The proposed approach searches for idle nodes and then tries to establish the fitness function of individual node to distribute the load. A GWO-based load-balancing method has been developed in (Patel, Patra, and B. Sahoo) for a containerized cloud system.

3. Container as a Service Cloud Architecture

CaaS cloud architecture shown in figure 2. includes the following components:

3.1. Container Orchestration Platform:

CaaS cloud architecture relies on the container orchestration platform. It automates container scaling, deployment by using container orchestration platforms such as Kubernetes, Docker Swarm, and Mesos are popular.

3.2. Virtual Machines:

A cluster of VMs hosts the container orchestration platform, which deploys containers. VMs isolate and secure host-based containers.

3.3. Resource Manager and Load Balancers:

The resource manager manages CaaS cluster resources. It guarantees enough virtual computers, storage, and other resources for cluster applications. The resource management monitors resource utilization and scales the cluster according to demand. The load balancer distributes traffic among several application instances in various containers. It keeps the app available and prepared to manage traffic.

3.4. Monitoring and Logging:

Any cloud architecture needs monitoring and logging, including CaaS. CaaS systems include monitoring and logging capabilities to help developers track application health and performance.

3.5. Container Mapper:

The container broker allocates containers to cluster virtual machines. Based on resources, application requirements, and other variables, it chooses a virtual machine to deploy a container. Container mapping optimizes resource use and eliminates virtual machine overload.

3.6. Container Scheduler:

The container scheduler schedules and manages CaaS cluster containers. It starts, monitors, and restarts containers. The container scheduler keeps programmes available and scalable.

4. Proposed System Model and Fitness Function Formulation

The proposed system model is comprehensively described in this section. It comprises five primary parts: the physical server model, the container

model, the virtual machine (VM) model, the task model, and the scheduling model.

4.1. System Model

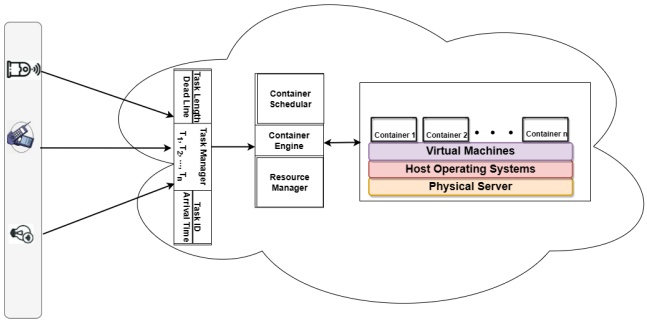

FIGURE 2. Container Scheduling Architecture in Cloud Environment

4.2. Container Model:

The model of a container based cloud system is defined by a collection of physical servers that offer the framework for producing containers through OS virtualization. To run an application, the containers obtain the necessary binaries and libraries from the host operating system running on the physical server. Hence, each container C_j is modelled by using three parameters based the configuration of physical server P_i. C ={$C_j{}^i$, memory$_j{}^i$, processing$_j{}^i$}, where $C_j{}^i$ denotes the jthcontainer on the ith server, memory$_j{}^i$ denotes the size of physical memory for jth container on the ith server, and processing$_j{}^i$ indicates the speed at which the jth container on the ith server processes data.

4.3. Virtual Machine Model:

A VM is a software emulation of a physical server. Let there be a set of virtual machines denoted by the VM = $\{vm_1, vm_2, \ldots, vm_l\}$, where each virtual machine vm1 creates a virtualized environment with its own operating system (OS), which can run multiple containers on top of the underlying hardware. Each VM is isolated from the other VMs running on the same physical machine with its own virtualized hardware resources (such as CPU, memory, storage, and network) that are allocated from the host machine's pool of resources. A VM is a server instance that runs a container.

4.4. Scheduling Model :

It is the job of the scheduling model in container orchestration platforms to distribute containers to the available physical servers based on how those servers are currently utilizing their resources and any other applicable constraints. The scheduling strategy makes certain that the containers are dispersed throughout the servers in a manner that allows for the highest possible level of resource utilization while also reducing the likelihood of an interruption in service.

4.5. Task Model :

The set of tasks, denoted by T = $\{t_1, t_2, .., t_n\}$ is composed of n tasks that arrive in real time from n users. In T, each task t_k has four properties, Task Id (T_{id}), Arrival Time (T_{arr}), Task Size (T_s) and Deadline (T_{dl}). Due to the task heterogeneity, the execution time T^{ij}_{kexec} of the kth task on the jth container of the ith server can be computed as in eq. (1).

$$T^{ij}_{kexec} = \frac{T_{sk}}{processing^j_i} \tag{1}$$

The total completion time $T^{ij}_{kcompletion}$ of the kth task on the jth container of the ith server can be computed as in eq. (2).

$$T^{ij}_{kcompletion} = T_{kwating} + T^{ij}_{kexec} \tag{2}$$

Where, $T_{kwating}$ stands for the total waiting time of the kth task.

Therefore, makespan, which measures the sum of all the completion times for all the tasks, is computed as in eq. (3).

$$makespan = \sum_k^n T_{kcompetion} \tag{3}$$

Server load variance measures how spread out the server load data is from the mean load. It is a measure of the average distance between each load reading and the mean load and is calculated by summing the squared differences between each induvial server load and the mean load of the server, and then dividing by the number of servers as shown in eq. (4)

$$P_{loadvar} = \frac{\sum_{i=1}^{n} (P_{iload} - \bar{P_{iload}})^2}{n} \tag{4}$$

Where, $P_{loadvar}$ is the load variation of all the physical server and P_{iload} is the load of ith physical server. $\bar{P_{load}}$ is the mean load of the server and n is the total number of servers.

4.6. Fitness Function

In general, a fitness functions is comprise of several Objective functions as: F (x) = $\{f_1$ (x), f_2 (x), . . . , f_n (x)$\}$, where f_n (x) represents n^{th} objective function.

• The first objective is to minimize the makespan as given in eq. (5)

$$f_1(makespan) = min(makespan) \tag{5}$$

• The second objective (eq. (6)) is to minimize the workload variance of the physical server because, the variance is used to assess the level of variability in the server load, and a higher variance indicates a greater level of variability and unpredictability in the server load. Therefore,

$$f_2(P_{loadvar}) = min(P_{loadvar}) \tag{6}$$

The fitness function can be defined as in eq. (7):

$$F = w1 * f\_1\ (makespan) + w2 * f\_2\ (P\_loadvar) \tag{7}$$

Where, w_1 and w_2 are weight factor and w_1 + w_2 = 1.

5. Honey Bee Mating Algorithm for Container Scheduling

Honey bees are social insects that can only thrive in a colony. The honey bee colony consists of three distinct types of individuals: the queen (the " breeding female"), the drones (the "males"), and the workers (the "nonbreeding females") (R. R. Sahoo et al.). Each individual performs a specific task for the colony and has developed unique instincts to meet the demands of that task. The HBMO Algorithm incorporates a number of distinct phases, and Fig. 3 illustrates the primary steps that are involved in the HBMA process. Probabilistically, drone and a queen mate using an annealing function as given eq. (8):

$$prod(D) = exp(-D(f)/S(t)) \tag{8}$$

where Prob (D) is the probability of a successful mating when drone D's sperm are added to the queen's spermatheca, D(f) is the absolute difference between drone D's fitness and the fitness of the queen, and S(t) is the momentum of the queen at time t. The fitness of each bee is calculated by using

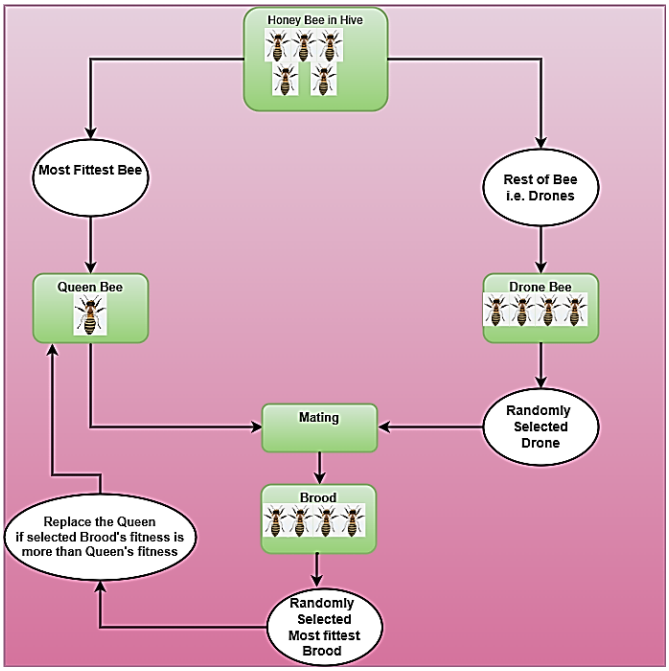

FIGURE 3. Honeybee mating procedure

eq. (7). When the queen is in it's highest speed or when the drone has the same degree of fitness as the queen, the chance of mating is high. The queen's momentum decreases after every spatial transition according to the eq. (9) and eq. (10):

$$S(t+1) = k \times S(t) \tag{9}$$

$$E(t+1) = m \times E(t) \tag{10}$$

where k and m are factors in the range [0, 1] denoting the relative speeds and energies dissipated at each stage of the process. Initially, the queen's speed is generated at random. Several mating flights are accomplished. Drones are generated at random at the starting of a mating flight, and the queen chooses one of them according to the eq. (8). If a drone and a queen successfully mate, the drone's sperm is kept in the queen's spermatheca (i.e., the drone meets the probabilistic decision criteria). The drone and the queen's genotypes are crossed to produce a new brood, that can be enhanced utilizing a workforce engaged in local search. Workers in the wild only take care of the young, so they aren't counted as individuals in the population count; instead, they are put to use in local search methods designed to boost the quality of the broods born from the queen's mating flights. If the new brood is superior to the present queen, it will replace her. If the brood is

unable to produce a new queen, one of its members will serve as a drone during the subsequent queen's mating flight.

Algorithm Honey Bee Mating Algorithm

Input: Nc, Nv, Nt, H, pop_size
Output: Best Solution
//Initialization of Parameters
H ← Number of Honey Bees in Population
Pop_size ← Population Size
Q ← Queen Bee
N_{itr} ← Number of Iteration
N_c ← Number of container to Schedule
N_t ← Number of Tasks
N_v ← Number of Virtual Machines
Fitness_val ← Fitness value
D ← Number of Drones
B ← Number of Broods
W ← Number of worker Bees
S_{max} ← Queen's highest Speed at the beginning of the mating flight
S_{min} ← Queen's lowest speed
E_{max} ← Queen's highest Energy
E_{min} ← Queen's lowest Energy
K ← Reduction rate of Energy
m ← Reduction rate of Speed
1. for i ← 1 to H do
2. Pop ← create_chromosome (Nc, Nv, Nt)
3. end for
4. for i ← 1 to N_{itr} do
5. Fitness_val ← fitness(Pop)
6. Choose the most fit bee as Queen &
set fitness_Q =fitness_Best_Bee;
7. for Iter =1 to N_{itr} do
8. begin
9. while energy >= E_{min} and Q's Spermathica is NOT full do
10. begin
11. choose a drone based on the Probability given in eq. (8);
12. keep the drone in spermatheca of Q;
13. ..update the energy and speed of queen as per (9) and (10);
14. end while
15. for i=1 to B do
16. begin
17. $Brood_i$=$Drone_i$+rand(0,1)*(Q -$Drone_i$);
18. Enhance the broods by local search;
19. end for
20. Choose the fittest as best_brood;
21. if fitness_best_brood > fitness_Q then
22. Q = fitness_best_brood;
23. end if
24. end for
25. return Q;// Best Solution

6. Simulation Results and Discussion

The Table 1 contains the list of parameters that were taken into consideration to carry out our experiment. All of the tests were performed in a rigid setting. The experiment is conducted with varied numbers of containers and tasks, ranging from 10 to 40 containers and 500 to 2000 tasks, respectively. After 500 repetitions, the load variance and makespan both remain constant. Consequently, the number of iterations is set to 500 for the experiments. The initial population is 500. We varied different parameters given in Table 1 and observed our experiment.

TABLE 1. Testbed Setting Parameters

Model	Parameters	Values
Cloud System	Number of Tasks	500-2000
	Task Size (MI)	3000 to 5000
	Arrival Time of Tasks	0 to 500 unit
	Deadline of Tasks	Arrival Time + U (2,15)
	Number of Virtual Machine (VM)	10 – 20
	Number of Containers	10, 20, 30, 40
	Execution rate of containers (MIPS)	2000-5000
HBMA	No. of Iteration	500
	Population Size	1000
	No. of Drones	999
	No. of Broods	500
	Capacity of Spermathica	999
	Momentum of Queen bound	50-100
	Energy of Queen bound	50-100

It is found that almost all tasks were completed within their assigned deadline. Our experiment includes server load variation and makespan. Makespan is the overall time required to accomplish all tasks. Low load variance indicates that even workload is evenly distributed among all the physical servers. The proposed Honey Bee Mating Algorithm outperformed all three techniques utilizing these two parameters.

Figure 4. depicts the variance of load among different physical servers with 10, 20, 30 and 40 numbers containers and a varying number of tasks from 500 to 2000. It can be observed from figure 4 that the proposed HBMA based model outperforms the GWO, PSO and GA based models in terms of variation in server load.

Figure 5. shows the makespan of the different sets of tasks varying from 500-2000 with 10, 20, 30 and 40 numbers of containers. Further, it can be observed from figure 5. that the proposed HBMA based model outperforms the GWO, PSO and GA based models in terms of makespan.

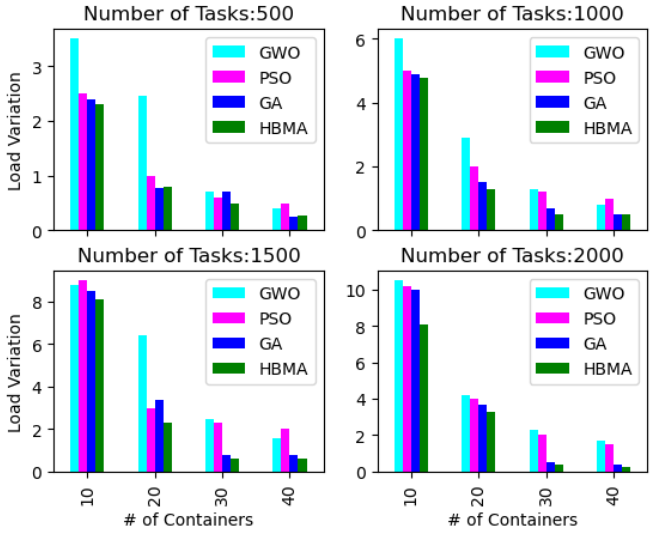

FIGURE 4. Workload variation with 500,1000,1500,2000 numbers of Tasks

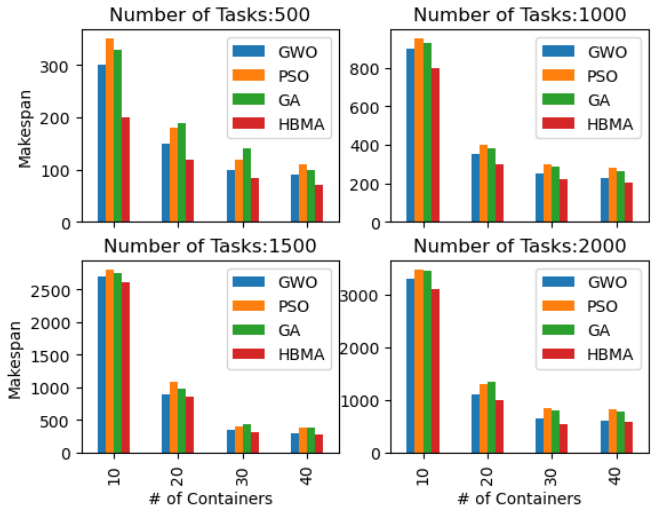

FIGURE 5. Makespan with 500,1000,1500, 2000 numbers tasks

7. Conclusion

Distributing the workload evenly across all servers while keeping the makespan to a minimum is the primary emphasis of the work discussed in this paper. To address the issue of load balancing in the containerized cloud, a Honey Bee Mating based optimization is proposed. The entire simulation is carried out with four different task sizes ((500, 1000, 1500 and 2000) and container counts (10, 20, 30, 40). Further, taking into account the parameter load variance and makespan, we compared our findings to those of the GA, PSO and GWO based methods. With regards to load variation and makespan, the experimental results show that the proposed HBMA based method outperforms all other methods.

Authors' Note

The authors of this article disclose no conflict of interest

References

Alguliyev, R M, Y N Imamverdiyev, and F J Abdullayeva. "PSO-based Load Balancing Method in Cloud Computing". *Automatic Control and Computer Sciences* 53.1 (2019): 45–55.

Babbar, Himanshi, et al. "A genetic load balancing algorithm to improve the QoS metrics for software defined networking for multimedia applications". *Multimedia Tools and Applications* 81.7 (2022): 9111–9129.

Dimitri, N. "Pricing cloud IaaS computing services". *Journal of Cloud Computing* 9.1 (2020): 14–14.

Elhoseny, A Francis ; Mohamed, et al. "Hybridization of firefly and Improved Multi-Objective Particle Swarm Optimization algorithm for energy efficient load balancing in Cloud Computing environments". *Journal of Parallel and Distributed Computing* 142 (2020): 36–45.

Gawali, Mahendra Bhatu and Subhash K Shinde. "Task scheduling and resource allocation in cloud computing using a heuristic approach". *Journal of Cloud Computing* 7.1 (2018): 4–4.

Hussein, Mohamed K, Mohamed H Mousa, and Mohamed A Alqarni. "A placement architecture for a container as a service (CaaS) in a cloud environment". *Journal of Cloud Computing* 8.1 (2019).

Maenhaut, Pieter-Jan J, et al. "Resource Management in a Containerized Cloud: Status and Challenges". *Journal of Network and Systems Management* 28.2 (2020): 197–246.

Niyogi, A Kishor; R., ; A Chronopoulos, and ; A Zomaya. "Latency and Energy-Aware Load Balancing in Cloud Data Centers: A Bargaining Game Based Approach". *IEEE Transactions on Cloud Computing* (2021).

Patel, Dimple, Manoj Kumar Patra, and Bibhudatta Sahoo. "GWO Based Task Allocation for Load Balancing in Containerized Cloud". *2020 International Conference on Inventive Computation Technologies (ICICT)* (2020): 655–659.

Patra, Manoj Kumar, et al. "GWO-Based Simulated Annealing Approach for Load Balancing in Cloud for Hosting Container as a Service". *Applied Sciences* 12.21 (2022): 11115–11115.

Pradhan, Arabinda and Sukant Kishoro Bisoy. "A novel load balancing technique for cloud computing platform based on PSO". *Journal of King Saud University - Computer and Information Sciences* 34.7 (2022): 3988–3995.

Sahoo, Rashmi Ranjan, et al. "A bio inspired and trust based approach for clustering in WSN". *Natural Computing* 15.3 (2016): 423–434.

Sanshi, Shridhar ; D, Pramodh Krishna D., and Ramesh Vatambeti. "Enhancing the Communication of IoT Using African Buffalo Delay Tolerant and Risk Packet Jump Approach". *WSEAS TRANSACTIONS ON INFORMATION SCIENCE AND APPLICATIONS* 19 (2022): 193–201.

Sefati, Seyedsalar, Maryamsadat Mousavinasab, and Roya Zareh Farkhady. "Load balancing in cloud computing environment using the Grey wolf optimization algorithm based on the reliability: performance evaluation". *The Journal of Supercomputing* 78.1 (2022): 18–42.

Wang, Bei and Jun Li. "Load balancing task scheduling based on Multi-Population Genetic Algorithm in cloud computing". *2016 35th Chinese Control Conference (CCC)*. IEEE, 2016. 5261–5266.

Wang, Tingting, et al. "Load Balancing Task Scheduling Based on Genetic Algorithm in Cloud Computing". *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 2014. 146–152.

Embargo period: The article has no embargo period.

To cite this Article: , Kodanda Dhar Naik, Rashmi Ranjan Sahoo , and Sanjay Kumar Kuanar . "**A Bio inspired Approach for Load Balancing in Container as a Service Cloud Computing Model**." International Research Journal on Advanced Science Hub 05.05S May (2023): 426–433. http://dx.doi.org/10.47392/irjash.2023.S058