International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 03 | Mar 2022 www.irjet.net p-ISSN: 2395-0072

A Host Selection Algorithm for Dynamic Container Consolidation in Cloud Data Centres using Particle Swarm Optimization

Prof.Radwan Saleh Dandah[1], Dr.Kasem Kabalan[2], Eng.Hayyan Rajab[3]

[1] *Professor, Department of Systems and Computer Networks, Faculty of Information Engineering, Tishreen University, Lattakia, Syria*
[2] *Assistant professor, Department of Systems and Computer Networks, Faculty of Information Engineering, Tishreen University, Lattakia, Syria.*
[3] *Postgraduate student (Phd), Department of Systems and Computer Networks, Faculty of Information Engineering, Tishreen University, Lattakia, Syria.*

Abstract - One of the leading causes of excessive energy consumption in cloud data centers is inefficient resource use. To address the issue, researchers proposed the Dynamic Container consolidation approach, which aims to consolidate containers into the fewest number of VMs and hosts. In this research, we introduce a novel host selection policy for container consolidation called the Energy Efficient Particle Swarm Optimization (EE-PSO) Algorithm to reduce energy consumption while maintaining the required performance levels in a cloud data center. We performed experimental evaluations using the ContainerCloudSim toolkit to validate the proposed algorithm's effectiveness with real-world workloads. The simulation results show that our proposed algorithm outperforms existing works in terms of energy consumption, QoS guarantees, number of newly created VMs, and number of container migrations.

Key Words: **Cloud Computing, Container as a Service (CaaS), Energy Efficiency, Dynamic Container Consolidation, Particle Swarm Optimization.**

1.INTRODUCTION

As a result of the rise of web-based applications such as micro-services and server-less architectures, containers have become more popular for creating an isolated, low overhead environment for deploying applications [1]. Container is an operating system-level virtualization that offers various advantages over virtual machines, including lightweight, mobility, low start-up time, and low resource usage [2]. Thus, containers might be seen as a new revolution in the cloud era and have been adopted from many cloud providers. Container as a service (CaaS) is the new service model that has been introduced in addition to traditional cloud services, including software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS).

Containers can be deployed either on physical machines or on virtual machines. Although deploying containers without the overhead of the hypervisor achieves high performance levels, there are some limitations for using this model such as: the dependency between containers and operating system type, suffering from security threats due to the fact that containers do not provide the same level of isolation as VMs [3]. Consequently, many cloud providers use a two-level virtualization architecture as shown in Fig -1.

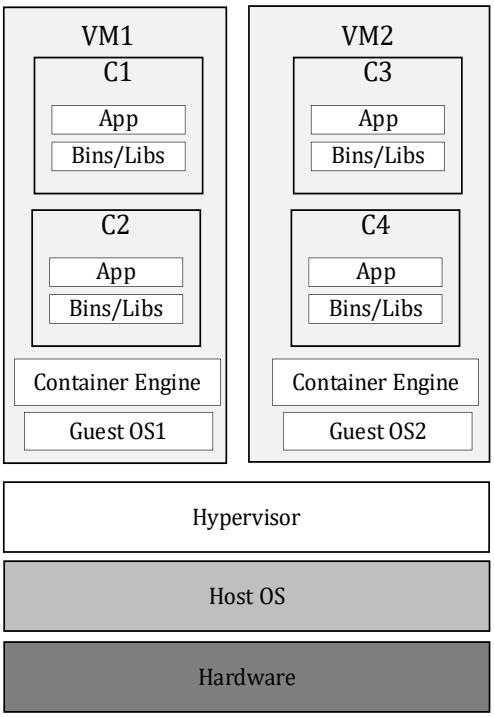

Fig -1: Two-level virtualization architecture [4]

In order to meet the increasing demand for cloud services, there has been a significant expansion in building data centers around the world, and statistics indicate that there are approximately 8.2 million data centers [5]. A

recently published study showed that data centers consumed about 205 TWh during 2018, which represents about 1% of global electricity consumption [6], and expectations indicate that this number could reach about 353 TWh by 2030 [7].

Servers are responsible for a large part of the power consumed in a cloud data center. Consequently, reducing the number of active servers, through dynamic consolidation of containers (or virtual machines), can significantly reduce power consumption while maintaining quality of service. The following issues should be able to be addressed by any container consolidation framework: [8]:

- When the host is detected as being overloaded and unable to provide the required resources for containers and virtual machines running on this host?
- Which containers should be selected to migrate from an overloaded host?
- When the host is identified as being underloaded? Is it possible to migrate all hosted containers and shut down this host?
- How to choose a destination (host/VM) for migrated containers?

According to the above questions, there are four sub-problems in dynamic container consolidation, in this paper we will focus on the sub-problem of destination host selection.

2. Related Work:

In contrast to the substantial study on energy efficiency of computing, for virtualized cloud data centers, only a few studies explored the challenge of energy efficient container management.

In [7], the researchers proposed a framework for container dynamic consolidation on virtual machines. They used static thresholds to determine the status of the hosts, Maximum Usage (MU) and Most Correlated (MCor) to select containers from overloaded hosts, First Fit (FF), least full (LF), and Correlation threshold (CorHs) to select a destination host for a migrated container, and finally, they used First Fit to place the migrated containers on a VM at the selected destination.

In [9], the researchers evaluated some of the container placement algorithms, including First Fit, Least Full, Most Full, and Random. These algorithms are used to select a running virtual machine on the host to be allocated to the container. The results showed that First Fit outperforms all the other algorithms in terms of energy consumption and number of migrations.

In [10], the authors showed the relationship between container and host selection policy. ACO, Least Full, Most Full, Most Correlated, Least Correlated, Max Variance, and Min Variance were considered for host selection, while for container selection, they considered max-correlated, max-variance, max-usage, and min-variance. They achieve a superior result using the maximum-variance host selection strategy and the greatest utilization container selection policy.

In [11], the researchers proposed MESF, or most efficient server first, which is a greedy container placement technique that assigns containers to the most energy efficient computers first. The suggested MESF method may greatly improve energy usage when compared to the Least Allocated Server First (LASF) and random scheduling schemes, according to simulation findings utilizing an actual set of Google cluster data as task input and machine set.

In [12], the researchers designed a fitness function to evaluate the resource wastage of VMs and Hosts, then they used Best Fit algorithm to create VMs in the hosts, and ACO to place containers on these VMs.

3. Background:

3.1 Data Center Power Model:

The power consumption of a data center at time t ($P_{dc}(t)$) can be calculated as the sum of power consumption of its servers at time t ($P_i(t)$) as shown in equation 1.

$$P_{dc}(t) = \sum_{i=1}^{N_s} P_i(t) \qquad (1)$$

where N_s is the number of servers. According to [13] there is a linear relationship between the CPU utilization and power consumption of the server. This relation can be formulated as follows:

$$P_i(t) = P_i^{idle} + \left(P_i^{max} - P_i^{idle}\right) * U_{i,t} \qquad (2)$$

where $U_{i,t}$ is the CPU utilization of server i at time t, P_i^{idle} and P_i^{max} represent the consumed power when the server is idle, or fully utilized respectively.

3.2 SLAV Metric:

Meeting QoS requirements is a very critical issue in cloud computing environment. These requirements can be formulated using several metrics such as: minimum throughput or maximum response time, but due to the fact that we do not have any prior knowledge about the behavior of the application running inside the container, it is important to identify a metric which does not depend on the workload. Researchers in [14] showed that the SLA can be violated if the virtual machine cannot get the required CPU which has been requested. Equation 3 shows how to calculate the SLAV metric:

$$SLAV = \sum_{i=1}^{N_s} \sum_{j=1}^{N_{vm}} \sum_{p=1}^{N_v} \frac{CPU_r(vm_{j,i},t_p) - CPU_a(vm_{j,i},t_p)}{CPU_r(vm_{j,i},t_p)} \qquad (3)$$

where N_{vm}, N_v are the number of VMs, the number of SLA Violations respectively, and $CPU_r(vm_{j,i}, t_p)$, $CPU_a(vm_{j,i}, t_p)$ are CPU amount requested and allocated by VM j on server i at the time t_p at which the violation p happened.

3.3 System Model:

In order to consolidate containers on the minimum number of virtual machines and hosts, we will use a similar model to the proposed model in [8], which consists of two modules: Host Status Module and Consolidation Module.

The Host Status Module exists in each active host, and it has three main components:

- Host Over-load/ Under-Load Detector: This component is responsible of deciding if the host is detected as being either over-loaded or under-loaded. In this research, the resource utilization is checked every 5 minutes using static thresholds to identify the status of the host.
- Container Selector: This component determines which containers should be selected to migrate from an overloaded host.
- Container Migration List: This list stores all containers selected by the Container Selector component.

The Consolidation Module runs on a separate host. This module is responsible for choosing a destination (host/VM) for migrated containers, and consists of the following components:

- Over-loaded Host List
- Over-loaded Destination Selector: This component uses a host selection policy to select a destination for a migrated container from an over-loaded host, and a container placement policy to choose one of the running VMs on that host to be allocated to the container. If there is no running vm to host a migrated container, the Vm Creator component is called.
- VM Creator: this component creates the largest possible vm on the host and assign the container to it.
- Destination List: This component stores the migration map decided by the over-loaded Destination Selector.
- Under-loaded Host List: This list contains all hosts that are identified as under-loaded after finding a destination for all migrated containers from over-loaded hosts.
- Under-loaded Destination Selector: This component tries to find appropriate destinations for all hosted containers by an under-loaded host. If this mission could be achieved, it sends the host ID to under-loaded host deactivator component.
- VM-Host Migration Manager: This component triggers the migration process after selecting all destinations.
- Under-loaded Host Deactivator: after migrating all containers of an under-loaded host, this component turns it off.

3.4 Standard Particle Swarm Optimization

PSO is a meta-heuristic algorithm that was introduced by Eberhart and Kennedy in 1995[15]. It mimics the social behavior of a flock of birds searching for food. Each bird inside the algorithm is called a particle, and each particle has a position vector that represents one of the possible solutions to the problem, as well as a velocity vector that describes its movement within the solution space. First, the position vectors are generated randomly. Second, at each iteration of the algorithm, each particle moves to a new position which depends on the values of the following: the particle's velocity in the previous iteration, the best position found by the particle (pbest); and the best position found in the entire swarm (gbest). The positions are evaluated using a fitness function. The following equations are used to calculate the new position and velocity vectors. [16]:

$$\overrightarrow{v_i^{t+1}} = w * \overrightarrow{v_i^t} + c_1 * rand_1 * \left(\overrightarrow{pbest_i} - \overrightarrow{x_i^t}\right) + c_2 * rand_2 * \left(\overrightarrow{gbest} - \overrightarrow{x_i^t}\right) \quad (4)$$

$$\overrightarrow{x_i^{t+1}} = \overrightarrow{x_i^t} + \overrightarrow{v_i^{t+1}} \quad (5)$$

where:

$\overrightarrow{v_i^t}$: velocity vector of particle i at iteration t

$\overrightarrow{v_i^{t+1}}$: velocity vector of particle i at iteration t+1

$\overrightarrow{x_i^t}$: current position of particle at iteration t

$\overrightarrow{x_i^{t+1}}$: new position of particle i at iteration t+1

$\overrightarrow{pbest_i}$: personal best position of particle i

$\overrightarrow{gbest}$: global best position

w: inertia weight

c_1,c_2: cognitive and social learning parameters respectively

$rand_1$,$rand_2$: random values between 0 and 1

Fig-2 shows the flowchart of the algorithm.

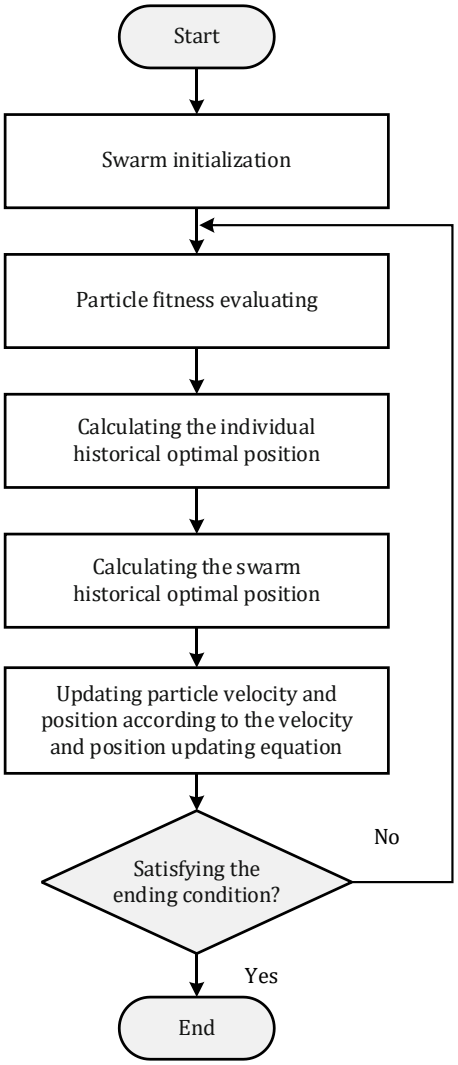

Fig-2: Flowchart of PSO [17]

4. Proposed Method:

4.1 Initialize Particles:

In the standard PSO, initial particles are generated randomly. However, this randomness reduces the algorithm's likelihood of converging to the optimal solution. As a result, effective initialization solutions can vastly increase its performance [18]. In our proposed method, we choose a random container from the migrated container list and assign it to an available host using the algorithm described in Fig-3. This approach ensures that the initialized solutions are good and feasible. The proposed host selection method is considered a modified version of the well-known bin-packing algorithm, First Fit. The hosts are arranged in descending order according to their power efficiencies and CPU utilization, respectively. The power efficiency of a host is a ratio between its CPU capacity and its maximum consumed power [19]. The algorithm examines hosts and their running VMs one by one. If there is a suitable vm and the host would not become overloaded after the allocation of the container, the selected host along with the selected vm are returned as the destination of the migrated container. If there is no running vm, the algorithm tries to create the largest possible VM on that host and assigns the container to it. It is important to note that creating VMs is only done in the process of finding destinations for migrated containers from overloaded hosts. The other difference between finding destinations for migrated containers from overloaded and underloaded hosts is the excluded hosts list, which includes the overloaded hosts in the first phase, and overloaded, switched off, and hosts which have been selected as destinations in the second phase.

Algorithm1: Destination Selection Process

Input: hostList,excludedHosts,container
Output: Destination(allocatedHost, allocatedVM)

```
allocatedHost ← NULL;
allocatedVM ← NULL;
hostList ← sort(hostList,PE,U,'Des');
foreach host in hostList do
    if excludedHosts.contains(host) then
    continue;
    end if
    vmList ← host.getVmList();
    foreach vm in vmList do
    if (vm.isSuitable(container) and
    host.isNotOverLoadedAfterAllocation(vm,container)) then
        allocatedHost ← host;
        allocatedVM ← vm;
        break;
    end if
    end foreach
    if (allocatedVM == NULL) then
    newVM = host.createLargestPossibleVM();
    if (newVM != NULL) then
        if (newVM.isSuitable(container) and
        host.isNotOverLoadedAfterAllocation(newVM, container)) then
            allocatedHost ← host;
            allocatedVM ← vm;
            break;
        end if
    end if
    end if
end foreach
return Destination;
```

Fig-3: Destination Selection Process

4.2 Fitness Function:

To formulate the fitness function of our proposed algorithm, we use a multi-criteria algorithm called TOPSIS [20]. According to this method, the best solution is the one which has the greatest distance from the negative-ideal solution and the smallest distance from the positive-ideal solution. There are four criteria depicted in table 1 used to rank the particles.

Table-1: Considered Criteria in Fitness Function

No	criteria	Description	Cost/benefit
1	Energy Consumption	The new power consumption of the data center after container migrations	Cost
2	Number of Successfully migrated containers	The higher this number, the higher the probability that the overloaded host will return to normal state, and the under-loaded host will be shut down	Benefit
3	The Sum of Energy Efficiency Factors of Selected Hosts	Selecting the most energy-efficient hosts has a great impact on energy consumption and performance	Benefit
4	Number Of Newley Created VMs	The lower this number, the lower overhead of launching a new operating system	Cost

First, we calculate the value of each parameter for every particle in the swarm. Then, these values are normalized by dividing them by the maximum value of each parameter found in the swarm using the equation:

$$Cri_{Norm}^{particle_i} = \frac{Cri_{particle_i}}{Cri_{max}} \quad (6)$$

In the next step, the Cri_{Norm}^{+} and Cri_{Norm}^{-} are determined according to the following equations:

$$Cri_{Norm}^{+} = Max\ Cri_{Norm}\ when\ CRI\ \in Benfit, and\ Min\ Cri_{Norm}\ when\ CRI\ \in Cost \quad (7)$$

$$Cri_{Norm}^{-} = Min\ Cri_{Norm}\ , when\ CRI\ \in Benfit\ and\ \ Max\ Cri_{Norm}\ when\ CRI\ \in Cost \quad (8)$$

Then, the Euclidean distances from the positive ideal solution $D_{particle_i}^{+}$ and the negative ideal solution $D_{particle_i}^{-}$ are calculated for each particle using the equation 9 and equation 10, respectively.

$$D_{particle_i}^{+} = \sum_{j=1}^{n} D(Cri_{Norm}^{particle_i}, Cri_{Norm}^{+}) \quad (9)$$

$$D_{particle_i}^{-} = \sum_{j=1}^{n} D(Cri_{Norm}^{particle_i}, Cri_{Norm}^{-}) \quad (10)$$

Finally, the score of the particle i is calculated using Equation 11:

$$Score_{particle_i} = \frac{D_{particle_i}^{-}}{D_{particle_i}^{+} + D_{particle_i}^{-}} \quad (11)$$

This score is regarded as the fitness function value that the algorithm seeks to maximize.

4.3 PSO Parameters:

The value of inertia weight w and its changes during the iterations of the algorithm are very critical to performance. In our proposed algorithm, we will use the approach introduced in [21], which leads to good results., as it depends on a linear reduction of w value for 70% of the iterations, and in the last stages, it gives w a random value within the range (0.4,0.7), which means greater values of w to allow the algorithm to jump outside the local optimal solution. The equation for calculating w is given as follows:

$$w = \begin{cases} w_{max} - \dfrac{t(w_{max} - w_{min})}{T} & t < 0.7 * T \\ 0.4 + 0.3 * rand() & t > 0.7 * T \end{cases} \quad (12)$$

Where t is the current iteration, T is the total number of iterations.

When a particle updates its position, the proposed algorithm checks the feasibility of the new solution. For simplicity, it examines the status of the selected host for each migrated container one by one, and if there is no suitable vm (running or newly created), the value of the corresponded element of the unallocated container at the position vector is set to null.

The rest of the algorithm parameters are shown in the Table -2.

Table-2: PSO Parameters

Number of particles	100
Initial inertia weight w	1.4
Minimum Value of w	0.4
Learning factors c1,c2	2
Number of iterations	100

5. Performance Evaluation:

5.1 Simulation Setup:

To evaluate the performance of our proposed policy, we will use the ContainerCloudSim toolkit [22]. In our experiments, the cloud datacenter consists of 100 PMs, 200 VMs, and more than 1000 containers. The characteristics of their configurations are shown in Table-3. PMs, VMs, and containers each have 1 TB, 2.5 GB, and 0.1 GB of disk storage. The network bandwidth of PMs, VMs, and containers is 1GB/s, 10MB/s, and 250KB/s, respectively. Because the startup delay of each container and VM creation directly affects the SLA measurements, these startup delays are significant and are set to 0.4 seconds for containers [8] and 100 seconds for VMs [23].

The CPU utilization of each container is assigned to one of PlanetLab workload traces [19]. These traces consist of 10 days of workload gathered every 5 minutes between March and April 2011 [9] as shown in Table-4.

(FFHS), Least Full Host Selection (LFHS), and Most Full Host Selection (MFHS). This comparison will be done in terms of energy consumption, SLAV, total number of migrations, and number of newly created VMs.

Table-4: PlanetLab Workload Traces

Date	Number of Containers	Mean(%)	St.dev(%)
2011/03/03	1052	12.31%	17.09%
2011/03/06	898	11.44%	16.83%
2011/03/09	1061	10.70%	15.57%
2011/03/22	1516	9.26%	12.78%
2011/03/25	1078	10.56%	14.14%
2011/04/03	1463	12.39%	16.55%
2011/04/09	1358	11.12%	15.09%
2011/04/11	1233	11.56%	15.07%
2011/04/12	1054	11.54%	15.15%
2011/04/20	1033	10.43%	15.21%

Table-3: Configuration of PMs, VMs, and containers

PM Configurations and power models				
PM type #	**CPU [3GHz] (mapped on 37274 MIPS Per core)**	**Memory (GB)**	**P^{idle}(Watt)**	**P^{max}(Watt)**
# 1	4 cores	64	86	117
# 2	8 cores	128	93	135
# 3	16 cores	256	66	247

Container and VM Types					
Container type #	**CPU MIPS (1 core)**	**Memory (MB)**	**VM type #**	**CPU [1.5 GHz] (mapped on 18636 MIPS Per core)**	**Memory (GB)**
# 1	4658	128	# 1	1 core	1
# 2	9320	256	# 2	2 cores	2
# 3	18636	512	# 3	4 cores	4
			# 4	8 cores	8

We use the First Fit algorithm as a container placement policy, and the Maximum Usage (MU) algorithm as a container selection policy.

The performance of our proposed algorithm (EE-PSO) is compared with the following algorithms: Correlation Threshold Host Selection (CorHS), First Fit Host Selection

5.2 Experiment Results:

5.2.1 Scenario 1:

In this set of experiments, the upper and lower thresholds are set at 80% ,70%, respectively. Because there are 10 days' workload data, each performance metric might

yield ten results. The average of these results is used as the final result of the algorithm on the metric. The results (Charts [1-4]) show that our proposed method outperforms all the other algorithms for all metrics because our algorithm aims to minimize the power consumption in its fitness function. EE-PSO also maximizes the number of successfully migrated containers, which means a lower number of overloaded hosts and SLA violations, and a higher number of underloaded hosts that will be shut down. On the other hand, selecting the most energy efficient hosts means not only minimizing the energy but also means higher capacities so the container will get its required resources and the number of migrations and created VMs will decrease.

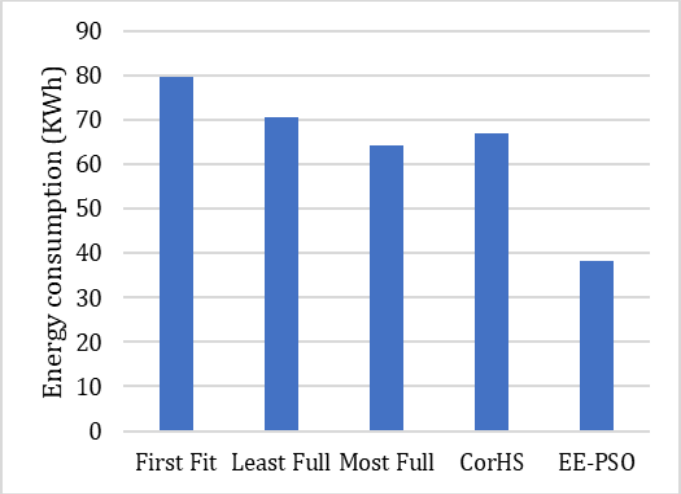

Chart -1: Energy consumption in scenario 1

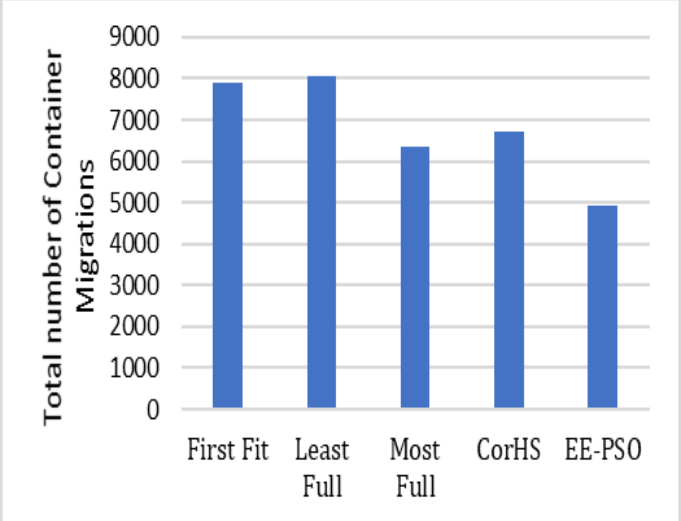

Chart -2: Total number of container migrations in scenario 1

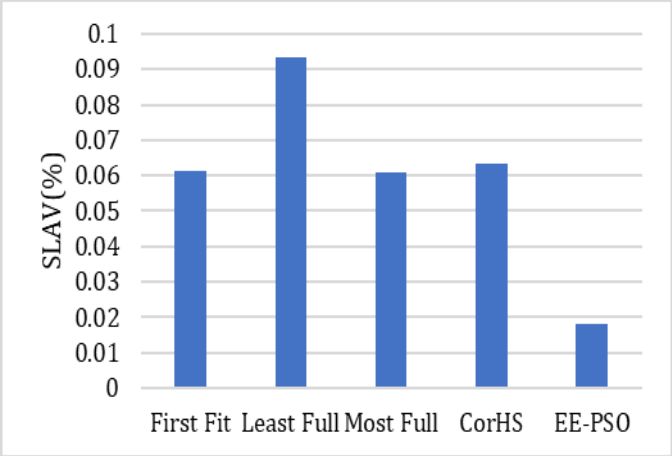

Chart -3: SLAV in scenario 1

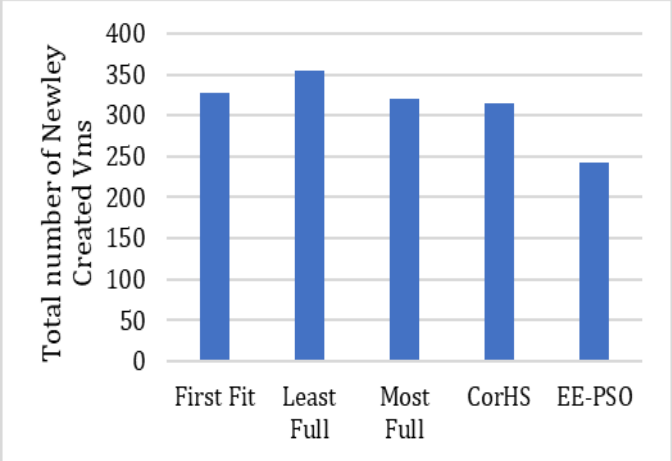

Chart -4: Total number of newly created VMs in scenario 1

5.2.2 Scenario 2:

In this set of experiments, we will study the effect of varying the upper-threshold value while keeping the lower threshold fixed at 70%. The number of containers and their workload traces are set according to day 1 of Table-4. The results (Charts [5-8]) show that increasing the upper-threshold value increases the power consumption metric for all algorithms because of the linear relationship between CPU utilization and energy consumed by the server. Also, when the upper-threshold is increased, the probability that a host would be identified as overloaded is decreased, which in turn decreases the number of containers selected to migrate, which results in a lower number of VM creations. On the other hand, the higher the upper-threshold, the more likely the host will not have adequate resources to adapt to fluctuations in container resource requirements, resulting in more SLA violations. At all upper-threshold values, our proposed algorithm performs better than all other algorithms.

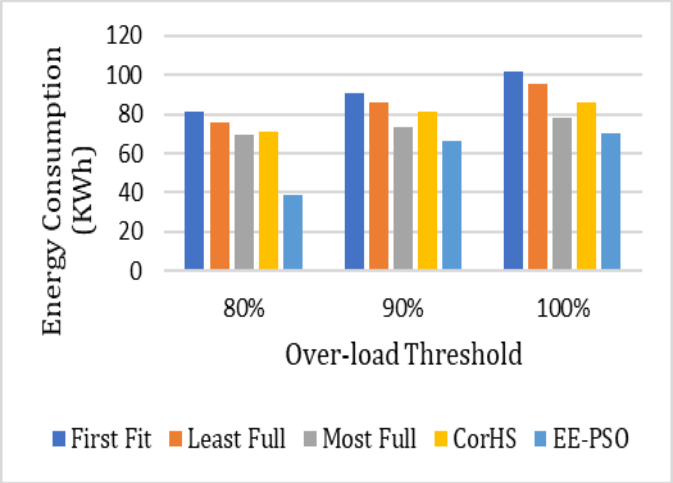

Chart -5: Energy consumption in scenario 2

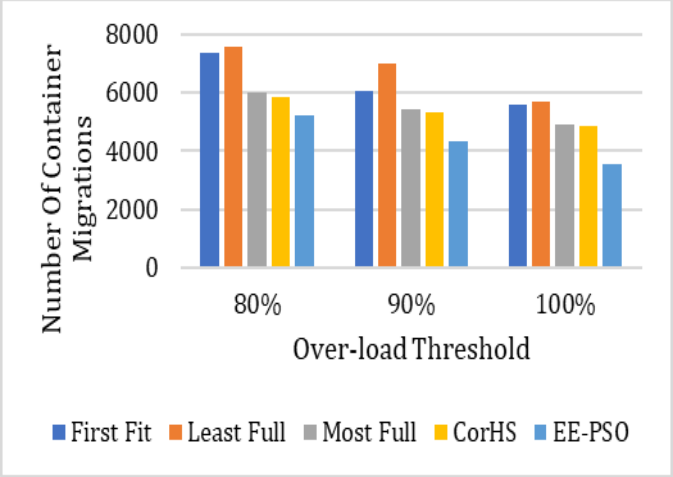

Chart -6: Total number of container migrations in scenario 2

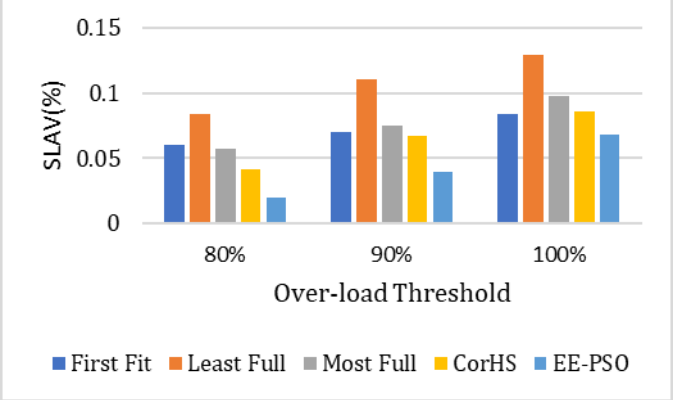

Chart -7: SLAV in scenario 2

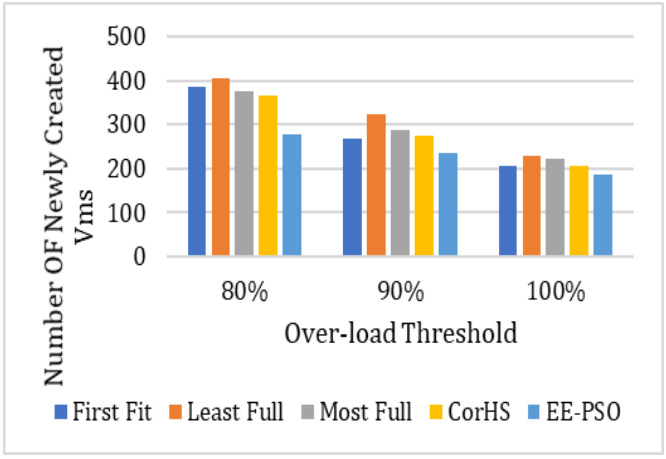

Chart -8: Total number of newly created VMs in scenario 2

5.2.3 Scenario 3:

In this set of experiments, the upper threshold is set at 80%, and the lower threshold will be varied. Also, the number of containers and their workload traces are set according to day 1 of Table-4. As shown in Charts [9-12], decreasing the lower-threshold increases the energy consumption metric since more hosts will be active, and the number of migrations will decrease because fewer hosts will be identified as underloaded. A lower container migration rate results in fewer VMs being created. Although the smaller number of created VMs when we decrease the lower-threshold, the size of these VMs will be bigger since we have more resources to allocate and the possibility that these VMs will not get their required resources in the future is high, which results in high SLA violations. The findings reveal that the performance of our proposed method outperforms that of competing algorithms across all metrics and lower-threshold values.

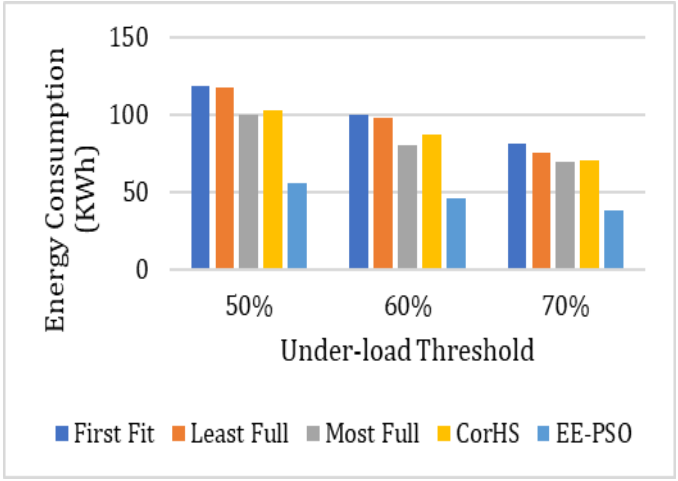

Chart -9: Energy consumption in scenario 3

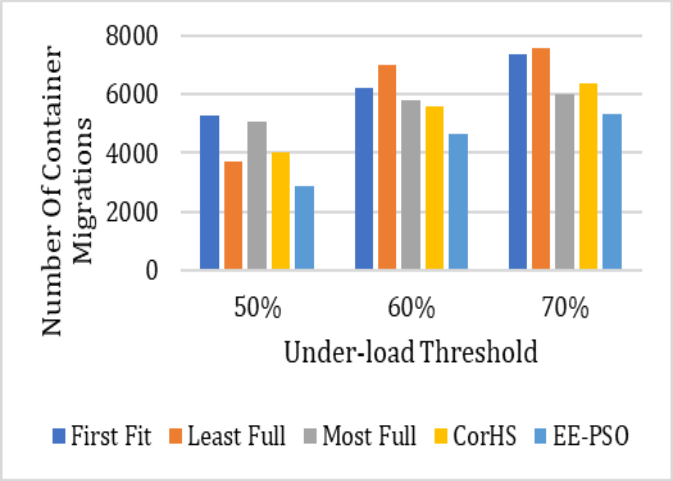

Chart -10: Total number of container migrations in scenario 3

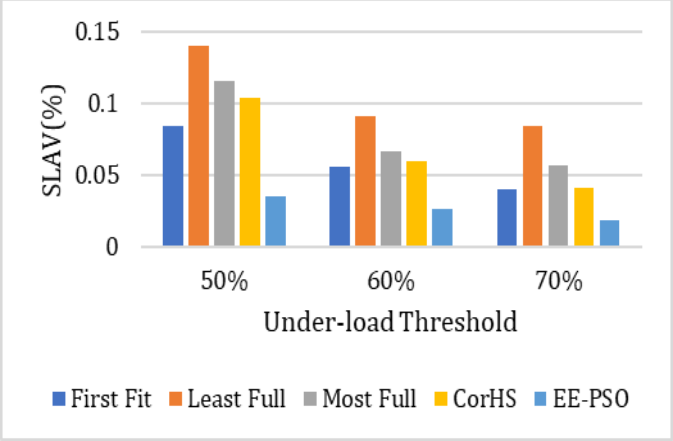

Chart -11: SLAV in scenario 3

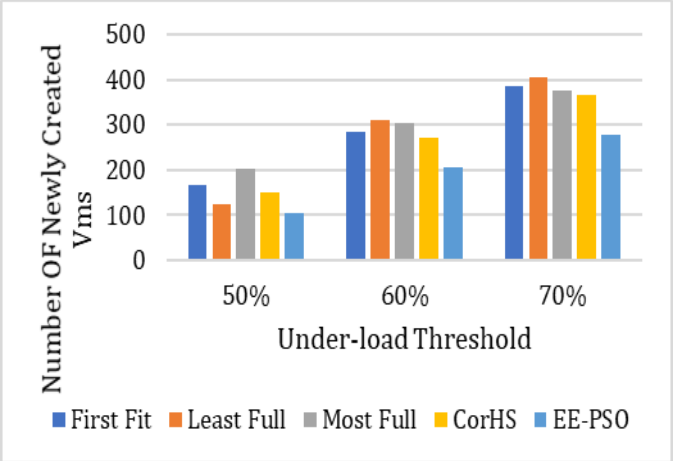

Chart -12: Total number of newly created VMs in scenario 3

6. CONCLUSION AND FUTURE WORK

Despite the growing popularity of Container as a Service (CaaS), the energy efficiency of resource management algorithms in this service paradigm has received little attention.

In this paper, we introduced A novel host selection algorithm for container consolidation by taking advantage of particle swarm optimization and energy efficiency of hosts. Three sets of simulation tests were performed to compare the performance of our approach with existing algorithms. Results show that our proposed method outperforms all competitive algorithms regarding energy consumption, total number of migrations, SLAV, and number of created VMs.

As for future work, we will improve our algorithm to make it aware of operating system type as a new constraint for the problem, and will extend our work to solve other container consolidation sub-problems.

REFERENCES

[1] B. Tan, H. Ma, Y. Mei, and M. J. I. T. o. C. C. Zhang, "A cooperative coevolution genetic programming hyper-heuristic approach for on-line resource allocation in container-based clouds," 2020.

[2] F. Paraiso, S. Challita, Y. Al-Dhuraibi, and P. Merle, "Model-driven management of docker containers," in *2016 IEEE 9th International Conference on cloud Computing (CLOUD)*, 2016, pp. 718-725: IEEE.

[3] O. Smimite and K. J. a. p. a. Afdel, "Container placement and migration on cloud system," 2020.

[4] C. Zhang, Y. Wang, H. Wu, and H. J. I. A. Guo, "An energy-aware host resource management framework for two-tier virtualized cloud data centers," vol. 9, pp. 3526-3544, 2020.

[5] B. K. Sovacool, C. G. Monyei, P. J. R. Upham, and S. E. Reviews, "Making the internet globally sustainable: Technical and policy options for improved energy management, governance and community acceptance of Nordic datacenters," vol. 154, p. 111793, 2022.

[6] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. J. S. Koomey, "Recalibrating global data center energy-use estimates," vol. 367, no. 6481, pp. 984-986, 2020.

[7] M. Koot and F. J. A. E. Wijnhoven, "Usage impact on data center electricity needs: A system dynamic forecasting model," vol. 291, p. 116798, 2021.

[8] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "A framework and algorithm for energy efficient container consolidation in cloud data centers," in *2015 IEEE International Conference on Data Science and Data Intensive Systems*, 2015, pp. 368-375: IEEE.

[9] P. Mishra, S. Bhatnagar, and A. Katal, "Cloud Container Placement Policies: A Study and Comparison," in

International Conference on Computer Networks and Inventive Communication Technologies, 2019, pp. 513-524: Springer

[10] W. A. Hanafy, A. E. Mohamed, and S. A. Salem, "Novel selection policies for container-based cloud deployment models," in *2017 13th International Computer Engineering Conference (ICENCO)*, 2017, pp. 237-242: IEEE.

[11] Z. Dong, W. Zhuang, and R. Rojas-Cessa, "Energy-aware scheduling schemes for cloud data centers on google trace data," in *2014 IEEE Online Conference on Green Communications (OnlineGreencomm)*, 2014, pp. 1-6: IEEE.

[12] M. K. Hussein, M. H. Mousa, and M. A. J. J. o. C. C. Alqarni, "A placement architecture for a container as a service (CaaS) in a cloud environment," vol. 8, no. 1, pp. 1-15, 2019.

[13] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. J. C. c. Jiang, "Power and performance management of virtualized computing environments via lookahead control," vol. 12, no. 1, pp. 1-15, 2009.

[14] A. Beloglazov, R. J. C. Buyya, C. Practice, and Experience, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," vol. 24, no. 13, pp. 1397-1420, 2012.

[15] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*, 1995, pp. 39-43: Ieee.

[16] F. Ebadifard, S. M. J. C. Babamir, C. Practice, and Experience, "A PSO-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment," vol. 30, no. 12, p. e4368, 2018.

[17] D. Wang, D. Tan, and L. J. S. C. Liu, "Particle swarm optimization algorithm: an overview," vol. 22, no. 2, pp. 387-408, 2018.

[18] S. Alsaidy, A. Abbood, and M. Sahib, "Heuristic initialization of PSO task scheduling algorithm in cloud computing," *Journal of King Saud University-Computer and Information Sciences,* 2020.

[19] F. F. Moges and S. L. J. J. o. C. C. Abebe, "Energy-aware VM placement algorithms for the OpenStack Neat consolidation framework," vol. 8, no. 1, pp. 1-14, 2019.

[20] M. Behzadian, S. K. Otaghsara, M. Yazdani, and J. J. E. S. w. a. Ignatius, "A state-of the-art survey of TOPSIS applications," vol. 39, no. 17, pp. 13051-13069, 2012.

[21] F. Luo, Y. Yuan, W. Ding, and H. Lu, "An improved particle swarm optimization algorithm based on adaptive weight for task scheduling in cloud computing," in *Proceedings of the 2nd International Conference on Computer Science and Application Engineering*, 2018, pp. 1-5.

[22] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, R. J. S. P. Buyya, and Experience, "ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers," vol. 47, no. 4, pp. 505-521, 2017.

[23] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 423-430: IEEE.