International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

ISSN:2147-6799 www.ijisae.org Original Research Paper

Load Balancing using Particle Swarm Optimization based Algorithms in Docker Container Cloud Environment: A Comparative Analysis

Manmitsinh Chandrasinh Zala [1], **Dr. Jaykumar Shantilal Patel** [2]

Submitted:14/03/2024 **Revised**: 28/04/2024 **Accepted**: 06/05/2024

Abstract*:* Cloud computing has vast usage in all type of services such as PaaS, SaaS, IaaS, XaaS , since last few years container based technologies have evolved and popular among industries and programmers, contrast with traditional Hypervisor based architecture container based applications are easy to load , deploy , secure and easy implementation , It also provides cluster based implementation and auto calling features, as of now multiple container based implantation is used in industries which leads to problem of resource allocation and efficient resource utilization , to maintain smooth and fair functioning of multiple containers over clusters load balancing mechanism is essential to distribute load equally to get maximum performance in cloud based services , Currently many technologies provides implementation of such as Nginx[18], kubernetes[14], and Docker Swarm[15] , here nginx and kubernetes provides default load balancing techniques , to improve this as per requirements many researchers have proposed various load balancing mechanisms. This paper is focused on comparison and result analysis of PSO (Particle Swarm Optimization) based algorithms proposed for load balancing in container based applications here we have showed and implemented various PSO algorithms for load balancing using parameters such as CPU usage, memory usage and optimize load allocation and finally concludes results comparisons of PSO algorithm variants..

Keywords: *Cloud Computing, Docker Container, Load balancing, Particle Swarm Optimization (PSO)*

1. Introduction

This In cloud computing based applications virtualization is used to facilitate hardware and software resources availability , this is useful to run virtualized applications over the shared resources , There are many challenges in cloud based architecture such as resource allocation ,security , efficient usage , privacy , availability and scaling, to provide virtualization there are mainly two fundamental technologies are being used 1.VM based virtualization(Hypervisor) and Container based technologies i.e. Docker, Kubernetes etc. the main difference between container based technologies and VM based technologies. (1) VM based virtualization and (2). Container based virtualization [15], VM-based virtualization uses a hypervisor to create and manage virtual machines (VMs), each running a full guest operating system and virtual hardware. This approach offers high isolation, strong security, and compatibility with multiple operating systems but incurs significant resource overhead, lower performance, and slower start up times due to the need to boot full OSes.[18] In contrast, container-based virtualization employs a container engine (like Docker) to manage containers that share the host OS kernel and run as isolated processes. Containers are lightweight, efficient, and start almost instantly, offering higher performance and portability across environments. However, they provide less isolation, posing potential security risks, are limited to applications compatible with the host OS, and may face resource contention. VMs are ideal for running diverse operating systems and applications requiring strong isolation, while containers are best for micro services, development environments, and applications needing efficient resource usage and rapid scaling [14] [15] 16].

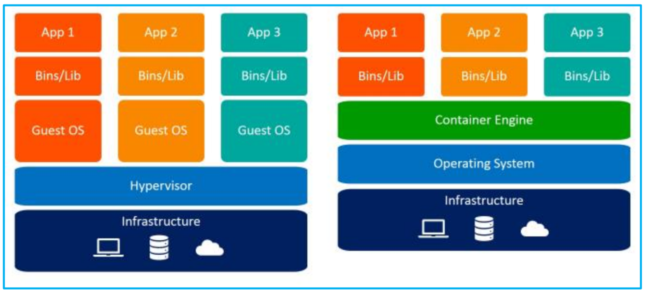

Fig. 1: VM based System Architecture and Container based System Architecture [14]

1 Gujarat Technological University, Ahmedabad, Gujarat-382350,, India
ORCID ID : 0000-0002-6987-2718
Manmit.zala@email.com
2 Chaudhari Technical Institute, Gandhinagar, Gujarat-382007, India
jay_sp_mca@yahoo.co.in

Table 1: Comparison of Virtual Machine (VM) and Container [14, 16, 18, 29, 32]

Feature	Virtual Machine	Container
OS	Requires same OS as client including Kernels, and other resources like CPU, Memory and Storage.	It works based on user mode kernels , so light weight and has required services inbuilt
Deployment	Takes time, restart is time consuming.	Easy to deploy
Fault Tolerance	Need to restart if gets failed	Can be easily created by an orchestrator if gets failed
Load Balancing	VM migration to different cluster is required	Container actually don't move Image (snapshot) is moved.

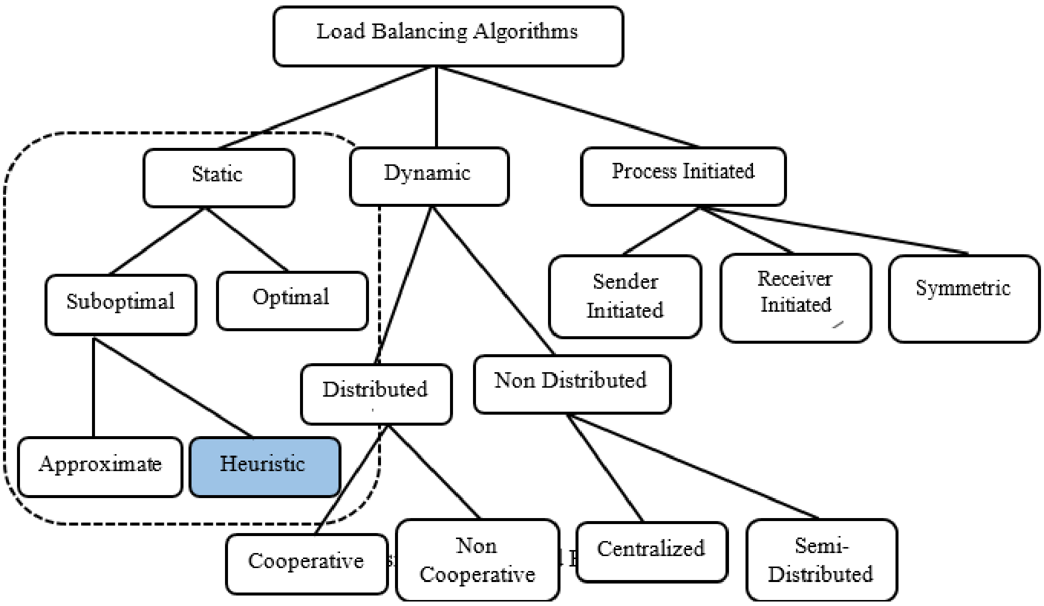

Fig 2 : Classification of Load balancing algorithms in cloud

1.1. Load Balancing

It is s a technique that distributes workload among various nodes in an environment to ensure no node is overloaded or idle at any given time. An effective load balancing algorithm ensures that each node performs a similar volume of work, improving response time and resource utilization. The algorithm maps incoming jobs to unoccupied resources, which is crucial in cloud computing due to the unpredictable number of requests. The primary goal is to dynamically allocate load among nodes to meet user requirements and maximize resource utilization. [17-21]

The core principle of load balancing is to distribute the workload evenly across all available nodes. This aims to enhance user satisfaction, which is increasingly important as user numbers and demands grow. An ideal load balancing algorithm optimally utilizes available resources, preventing nodes from being overloaded or under loaded. This process enables scalability, avoids bottlenecks, and reduces response time. Although many load balancing algorithms have been developed to distribute load Among various machines, achieving perfect load distribution remains an NP-complete problem, meaning no ideal algorithm currently exists that can allocate the load perfectly evenly across a system. [28], [42], [45]

Load balancing algorithms are essential for optimizing resource utilization and performance in distributed computing environments. Figure 2 shows these algorithms can be broadly classified into static and dynamic approaches. Static load balancing algorithms rely on a priori information about job characteristics, computing resources, and the communication network, making deterministic or probabilistic decisions at compile time that remain fixed during runtime. This approach is attractive due to its simplicity and minimal runtime overhead; however, it lacks responsiveness to dynamic runtime environments, potentially causing load imbalances and increased response times.

Conversely, dynamic load balancing algorithms leverage runtime state information to make real-time load-sharing decisions, providing robustness and flexibility suitable for modern systems. Dynamic algorithms can be further categorized based on several parameters: centralized versus decentralized, cooperative versus non-cooperative, adaptive versus non-adaptive, sender-initiated versus receiver-initiated, and pre-emptive versus non-pre-emptive. [20] Centralized algorithms gather necessary parameters via a single resource, advantageous when communication costs are low but prone to single points of failure and scalability limitations. Decentralized algorithms involve all resources in decision-making, enhancing scalability and fault tolerance. Cooperative algorithms involve distributed system components in collaborative decision-making, unlike non-cooperative algorithms.

[39] Adaptive algorithms adjust parameters during execution, in contrast to non-adaptive ones. In sender-initiated algorithms, overloaded nodes request process migration, while in receiver-initiated algorithms, under-loaded nodes initiate the request. Pre-emptive algorithms enable process transfer during execution, whereas non-pre-emptive algorithms consider only those processes awaiting CPU service.

Key functions of load balancing algorithms include load sensing, orchestration, balancing criteria calculation, task migration, and resource allocation requests, with actual load balancing occurring during task migration and decisions communicated to the Task Controller [45].

2. Related Work In Particle Swarm Optimization (Pso) Based Algorithm

2.1. Particle Swarm Optimization (PSO)

Introduced by Eberhart and Kennedy in 1995, PSO algorithm is based on bird flock food searching pattern, when bird flock is flying in search of food, they need to follow optimized pattern to land near the location of food as well as minimum risk of predators, all the birds follow the bird which has best position near the food.

Various Particle Swarm Optimization (PSO) variants have been developed to enhance load balancing in cloud computing environments, such as Docker. The standard PSO algorithm treats each particle as a potential solution, optimizing the distribution of Docker containers across nodes. Two-Memory PSO (TMPSO) employs two sets of memories for each particle, enhancing exploration and exploitation balance. Adaptive PSO dynamically adjusts parameters during optimization to accelerate convergence and avoid local minima. Multi-Objective PSO (MOPSO) handles multiple objectives simultaneously, using Pareto dominance for optimal solutions. Hierarchical PSO (HPSO) organizes particles hierarchically for improved exploration. Cooperative PSO (CPSO) involves multiple swarms cooperating to optimize different solution space parts. Discrete PSO (DPSO) is tailored for discrete optimization problems like container placement. Quantum-behaved PSO (QPSO) integrates quantum mechanics principles for better global search and faster convergence.

Hybrid PSO combines PSO with other techniques like Genetic Algorithms (GA) or Simulated Annealing (SA) for enhanced performance. Dynamic Multi-Swarm PSO (DMS-PSO) uses interacting swarms to adapt to dynamic environments. Opposition-based Learning PSO (OBL-PSO) enhances population diversity to escape local optima. Chaotic PSO (CPSO) utilizes chaotic maps to control parameters, preventing premature convergence. Constriction Factor PSO (CF-PSO) includes a constriction factor for convergence and stability. Table 2 provides a detailed comparison of these PSO variants

2.2. Equations and Functions of PSO:

2.2.1: Velocity Update

$$vi(t+1) = wvi(t) + c1r1(pi - xi(t)) + c2r2(g - xi(t)vi(t+1) = w\,vi(t) + c1\,r1\,(pi - xi(t)) + c2\,r2\,(g - xi(t))vi(t+1) = wvi(t) + c1r1(pi - xi(t)) + c2r2(g - xi(t)) \quad \textbf{(1)}$$

2.2.2: Position Update

$$xi(t+1) = xi(t) + vi(t+1)xi(t+1) = xi(t) + vi(t+1)xi(t+1) = xi(t) + vi(t+1) \quad \textbf{(2)}$$

2.2.3: Personal Best Update

$$\text{if } f\big(xi(t+1)\big) < f(pi) \text{ then } pi = xi(t+1)/ text\{if\ \} f(xi(t+1)) < f(pi)\ /text\{\ then\ \}\ pi = xi(t+1) if\ f(xi(t+1)) < f(pi)\ then\ pi = xi(t+1) \quad \textbf{(3)}$$

2.2.4: Global Best Update

$$if\ f(pi) < f(g)\ then\ g = pi\backslash text\{if\ \} f(p_i) < f(g)\backslash text\{\ then\ \} g = p_i\ \ if\ f(pi) < f(g)\ then\ g = pi \quad \textbf{(4)}$$

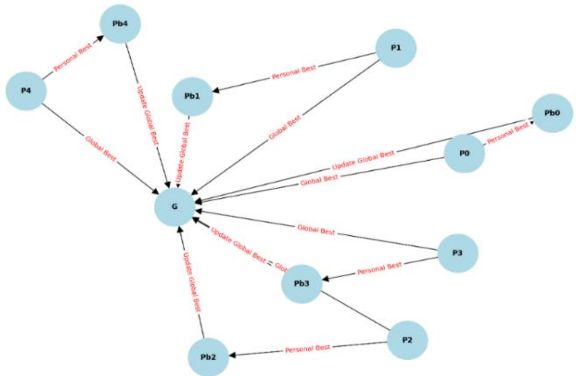

Fig 3: Graph representation of PSO algorithm

Figure 3 graph helps to visualize how particles in the PSO algorithm interact with their personal best positions and the global best position. We have generated figure 3 in python where node P0-P4 represents particles and Pb0- Pb4 represents personal best positions G is global best position and each particle tries to move related best positons indicated by arrows. In this paper we have implemented PSO, TMPSO, MOPSO and Adaptive PSO for load balancing and resource allocation for container based environment. Traditional PSO uses container resource allocation with most optimized solution, while TMPSO (Two Stage Multi option PSO has two steps **(I)** VM Selection **(ii)** VM Placement.VM selection uses first fit strategy while VM placement uses PSO operations to place the VM, While Adaptive PSO and MOPSO (Multi Objective Parallel Particle swarm optimization) uses combination of parallel PSO with micro service architecture for various requirement such as Computing storage, memory, failure rate etc.). We have also compared PSO, TMPSO with

Table 2: Comparison of various PSO based algorithms

Algorithm Name	Usage	Parameters	Result	Findings	Tools Used
Standard PSO [1]	Resource allocation, load balancing	Inertia weight, cognitive and social coefficients	Basic optimization, good convergence speed	Effective for simple problems, struggles with complex landscapes	MATLAB, Python, C++
Two-Memory PSO (TMPSO) [2]	Improved resource management	Inertia weight, cognitive and social coefficients	Enhanced optimization, faster convergence	Better memory utilization, more efficient than standard PSO	MATLAB, Python, Apache Bench
Adaptive PSO [3]	Scalability, fault tolerance	Adaptive control parameters, learning rates	Highly scalable, robust against faults	Balances performance and resource use, requires dynamic adjustment	MATLAB, Python, Java
Multi-Objective PSO (MOPSO) [4]	Handling multiple objectives	Multiple objective functions	Efficient multi-objective optimization, diverse solutions	High efficiency in multi-objective scenarios, suitable for complex problems	MATLAB, Python, R
Hierarchical PSO (HPSO) [5]	Task scheduling, data clustering	Hierarchical structure, social coefficients	Improved convergence, better resource utilization	Suitable for hierarchical problems, efficient clustering	MATLAB, Python, Java
Cooperative PSO (CPSO) [6]	Enhanced collaboration among particles	Cooperative parameters, social coefficients	Better convergence, enhanced optimization	Effective for problems requiring cooperation, improves overall performance	MATLAB, Python
Discrete PSO (DPSO) [7]	Task scheduling, data clustering	Position and velocity in discrete space	Efficient task scheduling, effective clustering	Suitable for discrete problems, limited by problem size	MATLAB, Python, C++
Quantum-behaved PSO (QPSO) [8]	Security optimization, energy efficiency	Quantum potential well, particle position	High security, low energy consumption	High efficiency in specific applications, requires careful tuning	MATLAB, Python, C#
Hybrid PSO [9]	Job-shop scheduling problem	Combination of PSO and other algorithms' parameters	Improved convergence, better resource utilization	Better performance in complex scenarios, more computationally intensive	MATLAB, Python, Java
Dynamic Multi-Swarm PSO (DMS-PSO) [10]	Dynamic resource allocation	Dynamic parameters, swarm division	Responsive to changing environment, efficient allocation	Adapts well to changing conditions, can be complex to implement	MATLAB, Python, R
Opposition-based Learning PSO (OBL-PSO) [11]	Global optimization	Opposition-based learning parameters	Enhanced global search capability, faster convergence	Balances exploration and exploitation, improves overall optimization efficiency	MATLAB, Python, C++
Chaotic PSO (CPSO) [12]	Resource allocation, load balancing	Chaotic sequences, acceleration coefficients	Improved convergence, better resource utilization	Utilizes chaos theory for optimization, can be more effective for complex landscapes	MATLAB, Python, Java
Constriction Factor PSO (CF-PSO) [13]	Resource allocation, load balancing	Constriction factor, cognitive and social coefficients	Stable convergence, avoids premature convergence	Improved stability and convergence, effective for a wide range of optimization problems	MATLAB, Python, C++

Adaptive PSO, adaptive PSO adapts the global best position and its balances the performance, resource use, but it requires higher adjustment due to dynamic nature.

3. Proposed Model And Experimental Setup

3.1. Proposed Model:

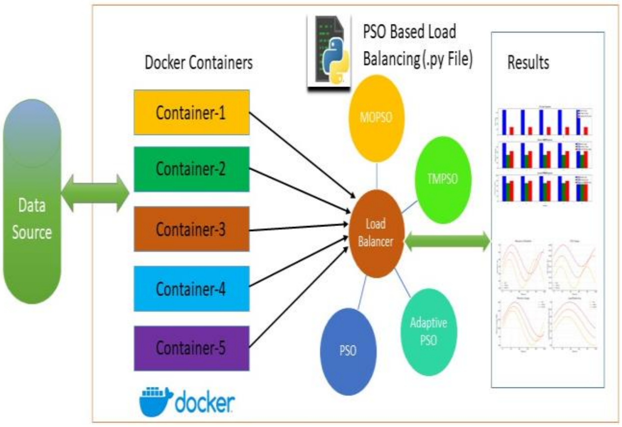

Fig 4: Proposed Model for Experimental setup

Table 3: Average Resource Utilization, CPU and Memory Utilization

Container Name	CPU Usage	Memory Usage	Total Resource Usage
PSO	50.93	100.18	51.86
TMPSO	55.85	105.31	56.71
MOPSO	60.22	111.9	60.45
PSO2	50	104.09	50
MOPSO2	60.24	116.33	60.49

Resource Utilization: Percentage of resources (CPU, memory, etc.) used over time.

CPU Usage: Percentage of CPU used over time.

Memory Usage: Amount of memory used over time.

Load Balancing: Effectiveness of load distribution over time demagnetizing factor

We have performed execution of request on each container for 100 times on 5 different containers where PSO and MOPSO requires 2 containers for the given load data, while TMPSO requires only 1 container for same operations.

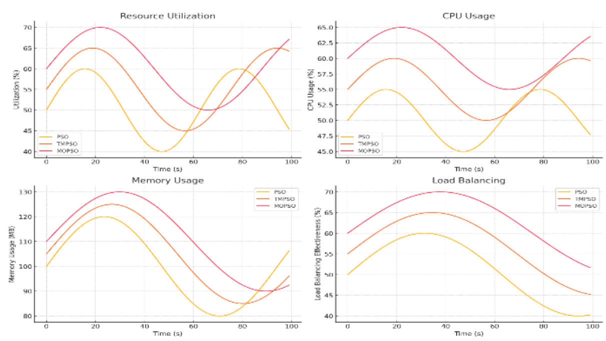

Figure 5: Resource Utilization, CPU Usage, Memory Usage, and Load Balancing using PSO algorithms.

Table 3 shows average CPU and memory usage also total resource utilization percentage, we can see the effect of load balancing using these three algorithms in figure 4,5,6, and 7 while figure 8 shows load balancing over a time PSO based algorithms have better load balancing output compare to other heuristic algorithms.

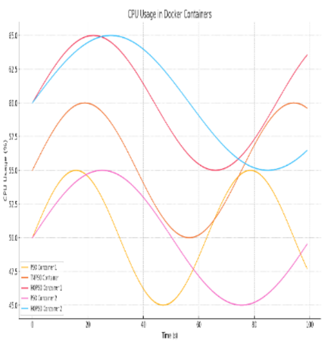

Figure 6: CPU Usage in Docker Container

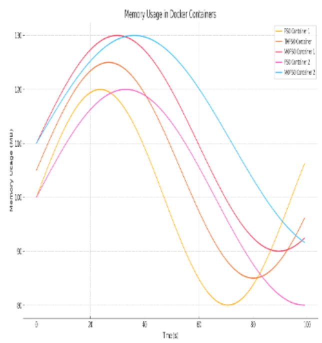

Figure 7: Memory Utilization in Docker Container

Figure 5,6 and 7 shows that Simple PSO requires less memory and CPU utilization compare to other variants but over a time for better load balancing results are produced by MOPSO and Adaptive PSO, initially they require more resources but later on they provide better performance compare to PSO and TMPSO.

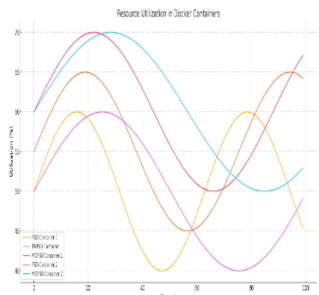

Figure 8: Load balancing Effectiveness over Time

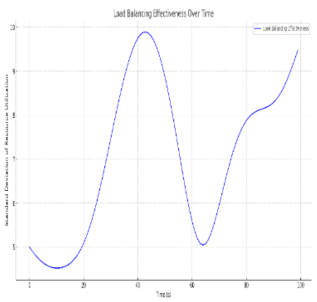

Figure 9: Load balancing Effectiveness over Time

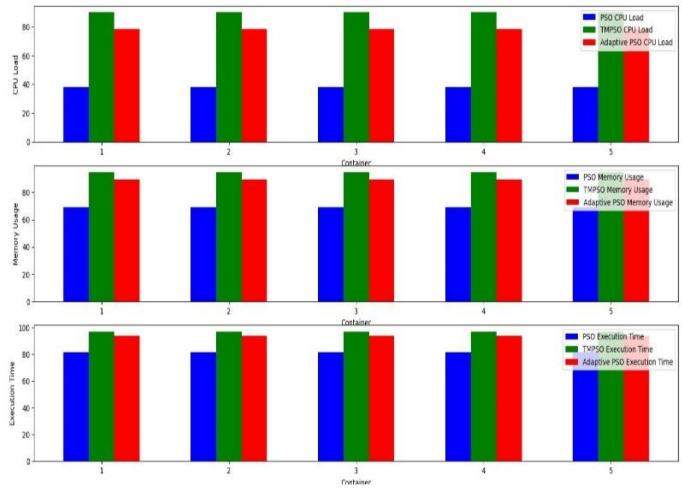

Figure 10: CPU Load Comparison using PSO Algorithm (PSO, TMPSO, and Adaptive PSO).

Table 4: Comparative Analysis of PSO, TMPSO and Adaptive PSO

Container	Algorithm	CPU Load	Memory Usage (MB)	Execution Time (Sec.)
1	PSO	23.5655	61.78275	77.06965
2	PSO	23.40568	61.70284	77.0217
3	PSO	23.3991	61.69955	77.01973
4	PSO	23.45902	61.72951	77.03771
5	PSO	23.46109	61.73055	77.03833
Average	**PSO**	**23.46**	**61.73**	**77.04**
1	TMPSO	50.18283	75.09142	85.05485
2	TMPSO	49.44382	74.72191	84.83315
3	TMPSO	49.27857	74.63929	84.78357
4	TMPSO	49.75902	74.87951	84.92771
5	TMPSO	49.61538	74.80769	84.88462
Average	**TMPSO**	**49.66**	**74.83**	**84.9**
1	Adaptive PSO	25.99327	62.99663	77.79798
2	Adaptive PSO	26.3002	63.1501	77.89006
3	Adaptive PSO	26.2197	63.10985	77.86591
4	Adaptive PSO	26.78923	63.39462	78.03677
5	Adaptive PSO	26.87113	63.43557	78.06134
Average	Adaptive PSO	**26.43**	**63.22**	**77.93**

4. Conclusion and Future Work

4.1. Conclusions

Container provides light weight virtualization and has better performance over traditional VMs; to analyse load balancing for each container, we need to define a metric that measures the effectiveness of load distribution. Load balancing metrics can include the standard deviation of CPU and memory usage across containers, which indicates how evenly the load is distributed. Since we already have hypothetical data for resource utilization, CPU usage, and memory usage, we can use this data to calculate load balancing effectiveness. We have followed below steps.

1. Calculate the standard deviation of CPU and memory usage across containers over time.

2. Plot these standard deviations to visualize load balancing effectiveness.

Based on the comparative analysis in Figure 8 , Figure 9 and Figure 11 of PSO, TMPSO, MOPSO, and Adaptive PSO algorithms, PSO demonstrated the best overall performance in terms of CPU load, memory usage, and execution time for the given single-objective problem. With an average CPU load of 23.46%, memory usage of 61.73 MB, and execution time of 77.04 seconds, PSO's simplicity resulted in efficient resource utilization and quick convergence. TMPSO, designed for dynamic environments, exhibited the highest average CPU load of 49.66%, memory usage of 74.83 MB, and execution time of 84.9 seconds, indicating significant computational overhead and suboptimal performance in this static, single-objective context.

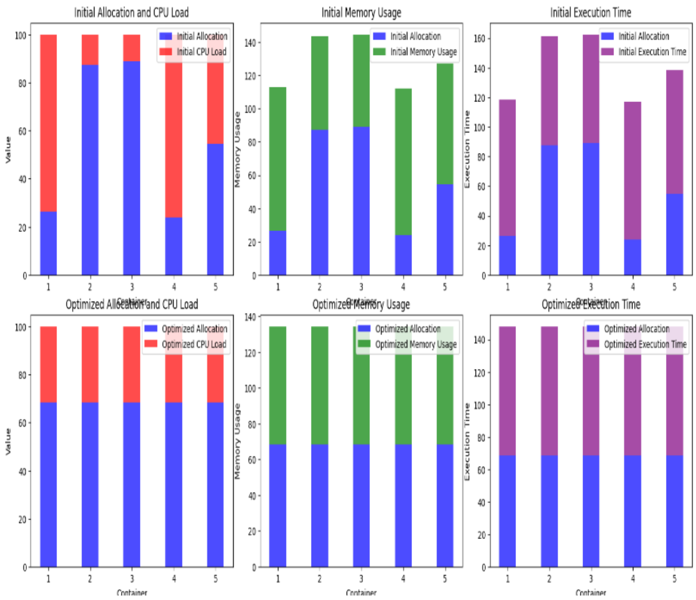

Fig. 11: Initial and balanced comparison of PSO based algorithms. (PSO, TMPSO, and Adaptive PSO).

Adaptive PSO showed balanced resource usage with an average CPU load of 26.43%, memory usage of 63.22 MB, and execution time of 77.93 seconds, benefiting from dynamic load distribution and efficient resource allocation. While MOPSO was not directly compared in the detailed data, it is generally known for higher resource consumption due to its multi-objective optimization focus, which may not be justified in single-objective problems. Overall, PSO's ease of tuning and well-understood parameters make it ideal for simple optimization tasks, while Adaptive PSO offers robust performance in dynamic scenarios. TMPSO's complexity may not provide significant advantages in static environments, and MOPSO should be reserved for complex, multi-objective tasks.

4.2. Future Work

In future we will try to implement more heuristic based algorithms for efficient resource utilization with less CPU time and memory requirement, above comparison shows that PSO based algorithms have effective mechanism for load balancing and container scheduling. More classification methods can be combined with PSO based algorithm to solve load balancing and resource allocation problem in container-based architecture.

Author contributions

Both Authors have contributed equally

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] Kennedy, J., & Eberhart, R. (1995). "Particle Swarm Optimization." Proceedings of IEEE International Conference on Neural Networks (Vol. 4, pp. 1942-1948). IEEE.

[2] Tang, Y., & Zhang, W. (2006). "Two-Memory Particle Swarm Optimization." Proceedings of the IEEE Congress on Evolutionary Computation (pp. 1861-1867). IEEE.

[3] Ratnaweera, A., Halgamuge, S. K., & Watson, H. C. (2004). "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients." IEEE Transactions on Evolutionary Computation, 8(3), 240-255.

[4] Coello, C. A., Pulido, G. T., & Lechuga, M. S. (2004). "Handling multiple objectives with particle swarm optimization." IEEE Transactions on Evolutionary Computation, 8(3), 256-279.

[5] Gazi, V., & Passino, K. M. (2004). "Stability analysis of social foraging swarms." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34(1), 539-557.

[6] Van den Bergh, F., & Engelbrecht, A. P. (2004). "A cooperative approach to particle swarm optimization." IEEE Transactions on Evolutionary Computation, 8(3), 225-239.

[7] Kennedy, J., & Eberhart, R. C. (1997). "A discrete binary version of the particle swarm algorithm." Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation (Vol. 5, pp. 4104-4108). IEEE.

[8] Sun, J., & Wu, X., Palade, V., & Fang, W. (2012). "Quantum-behaved particle swarm optimization: analysis of individual particle behavior and parameter selection." Evolutionary Computation, 20(3), 349-393.

[9] Zhang, X., Zhou, Y., & Jiao, L. (2008). "An improved particle swarm optimization algorithm for job-shop scheduling problem." International Journal of Advanced Manufacturing Technology, 38(7-8), 731-737.

[10] Liang, J. J., Qin, A. K., Suganthan, P. N., & Baskar, S. (2006). "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions." IEEE Transactions on Evolutionary Computation, 10(3), 281-295.

[11] Rahnamayan, S., Tizhoosh, H. R., & Salama, M. M. A. (2008). "Opposition-based differential evolution." IEEE Transactions on Evolutionary Computation, 12(1), 64-79.

[12] Li, X., & Yin, M. (2013). "A hybrid particle swarm optimization with sine cosine acceleration coefficients." Expert Systems with Applications, 40(1), 174-184.

[13] Clerc, M., & Kennedy, J. (2002). "The particle swarm-explosion, stability, and convergence in a multidimensional complex space." IEEE Transactions on Evolutionary Computation, 6(1), 58-73.

[14] B. Bashari Rad, H. John Bhatti, and M. Ahmadi, "An Introduction to Docker and Analysis of its Performance," *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 17, no. 3, pp. 228–235, 2017.

[15] J. Lv, M. Wei, and Y. Yu, "A container scheduling strategy based on machine learning in microservice architecture," *Proc. - 2019 IEEE Int. Conf. Serv. Comput. SCC 2019 - Part 2019 IEEE World Congr. Serv.*, pp. 65–71, 2019, doi: 10.1109/SCC.2019.00023.

[16] J. Bhimani, Z. Yang, M. Leeser, and N. Mi, "Accelerating big data applications using lightweight virtualization framework on enterprise cloud," *2017 IEEE High Perform. Extrem. Comput. Conf. HPEC 2017*, 2017, doi: 10.1109/HPEC.2017.8091086.

[17] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall, "An Empirical Analysis of the Docker Container Ecosystem on GitHub," *IEEE Int. Work. Conf. Min. Softw. Repos.*, pp. 323–333, 2017, doi: 10.1109/MSR.2017.67.

[18] H. Rajavaram, V. Rajula, and B. Thangaraju, "Automation of Microservices Application Deployment Made Easy By Rundeck and Kubernetes," *2019 IEEE Int. Conf. Electron. Comput. Commun. Technol. CONECCT 2019*, pp. 3–5, 2019, doi: 10.1109/CONECCT47791.2019.9012811.

[19] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Autonomic Vertical Elasticity of Docker Containers with ELASTICDOCKER," *IEEE Int. Conf. Cloud Comput. CLOUD*, vol. 2017-June, pp. 472–479, 2017, doi: 10.1109/CLOUD.2017.67.

[20] F. Wan, X. Wu, and Q. Zhang, "Chain-Oriented Load Balancing in Microservice System," *2020 World Conf. Comput. Commun. Technol. WCCCT 2020*, pp. 10–14, 2020, doi: 10.1109/WCCCT49810.2020.9169996.

[21] C. Singh, N. S. Gaba, M. Kaur, and B. Kaur, "Comparison of different CI/CD Tools integrated with cloud platform," *Proc. 9th Int. Conf. Cloud Comput. Data Sci. Eng. Conflu. 2019*, pp. 7–12, 2019, doi: 10.1109/CONFLUENCE.2019.8776985.

[22] G. Ambrosino, G. B. Fioccola, R. Canonico, and G. Ventre, "Container mapping and its impact on performance in containerized cloud environments," *Proc. - 14th IEEE Int. Conf. Serv. Syst. Eng. SOSE 2020*, pp. 57–64, 2020, doi: 10.1109/SOSE49046.2020.00014.

[23] Y. Xu and Y. Shang, "Dynamic Priority based Weighted Scheduling Algorithm in Microservice System," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 490, no. 4, 2019, doi: 10.1088/1757-899X/490/4/042048.

[24] N. NaiK, "Docker Container Based Big Data Processing System In Multiple Clouds for Everyone," vol. 29, no. 3, pp. 712–717, 2017, doi: 10.3788/AOS20092903.0712.

[25] H. Zeng, B. Wang, W. Deng, and W. Zhang, "Measurement and evaluation for docker container networking," *Proc. - 2017 Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discov. CyberC 2017*, vol. 2018-Janua, pp. 105–108, 2017, doi: 10.1109/CyberC.2017.78.

[26] Q. Li and Y. Fang, "Multi-algorithm collaboration

scheduling strategy for docker container," *2017 Int. Conf. Comput. Syst. Electron. Control. ICCSEC 2017*, pp. 1367–1371, 2018, doi: 10.1109/ICCSEC.2017.8446688.

[27] D. N. Jha, S. Garg, P. P. Jayaraman, R. Buyya, Z. Li, and R. Ranjan, "A holistic evaluation of docker containers for interfering microservices," *Proc. - 2018 IEEE Int. Conf. Serv. Comput. SCC 2018 - Part 2018 IEEE World Congr. Serv.*, no. VM, pp. 33–40, 2018, doi: 10.1109/SCC.2018.00012.

[28] Y. Kang and R. Y. C. Kim, "Twister Platform for MapReduce Applications on a Docker Container," *2016 Int. Conf. Platf. Technol. Serv. PlatCon 2016 - Proc.*, no. i, pp. 16–18, 2016, doi: 10.1109/PlatCon.2016.7456834.

[29] V. G. da Silva, M. Kirikova, and G. Alksnis, "Containers for Virtualization: An Overview," *Appl. Comput. Syst.*, vol. 23, no. 1, pp. 21–27, 2018, doi: 10.2478/acss-2018-0003.

[30] "Bowen Ruan, Hang Huang , SongWu ,andHaiJin " Performance Study of Conteiners In Cloud Environment.pdf." Springer International Publishing.

[31] M. Cerqueira De Abranches and P. Solis, "An algorithm based on response time and traffic demands to scale containers on a Cloud Computing system," *Proceedings - 2016 IEEE 15th International Symposium on Network Computing and Applications, NCA 2016*. pp. 343–350, 2016, doi: 10.1109/NCA.2016.7778639.

[32] M. Beranek, V. Kovar, and G. Feuerlicht, *Framework for Management of Multi-tenant Cloud Environments*, vol. 10967 LNCS. Springer International Publishing, 2018.

[33] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS," *Proc. - 2014 IEEE Int. Conf. Cloud Eng. IC2E 2014*, pp. 610–614, 2014, doi: 10.1109/IC2E.2014.41.

[34] P. Mohan, T. Jambhale, L. Sharma, S. Koul, and S. Koul, "Load Balancing using Docker and Kubernetes: A Comparative Study," *Int. J. Recent Technol. Eng.*, vol. 9, no. 2, pp. 782–792, 2020, doi: 10.35940/ijrte.b3938.079220.

[35] A. Khan, "Key Characteristics of a Container Orchestration Platform to Enable a Modern Application," *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 42–48, 2017, doi: 10.1109/MCC.2017.4250933.

[36] N. Nguyen and D. Bein, "Distributed MPI cluster with Docker Swarm mode," *2017 IEEE 7th Annu. Comput. Commun. Work. Conf. CCWC 2017*, 2017, doi: 10.1109/CCWC.2017.7868429.

[37] K. Ye and Y. Ji, "Performance Tuning and Modeling for Big Data Applications in Docker Containers," *2017 IEEE Int. Conf. Networking, Archit. Storage, NAS 2017 - Proc.*, 2017, doi: 10.1109/NAS.2017.8026871.

[38] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the Cloud," *Futur. Gener. Comput. Syst.*, vol. 68, pp. 175–182, 2017, doi: 10.1016/j.future.2016.08.025.

[39] M. Rusek, D. Rzegorz, and A. Orłowski, "A decentralized system for load balancing of containerized microservices in the cloud," *Int. Conf. Syst. Sci.*, vol. 539, no. November, pp. 142–152, 2016, doi: 10.1007/978-3-319-48944-5.

[40] E. Jafarnejad Ghomi, A. Masoud Rahmani, and N. Nasih Qader, "Load-balancing algorithms in cloud computing: A survey," *J. Netw. Comput. Appl.*, vol. 88, pp. 50–71, 2017, doi: 10.1016/j.jnca.2017.04.007.

[41] J. Cito and H. C. Gall, "Using docker containers to improve reproducibility in software engineering research," *Proc. - Int. Conf. Softw. Eng.*, vol. 1, pp. 906–907, 2016, doi: 10.1145/2889160.2891057.

[42] C. Cérin, T. Menouer, W. Saad, and W. Ben Abdallah, "A New Docker Swarm Scheduling Strategy," *Proc. - 2017 IEEE 7th Int. Symp. Cloud Serv. Comput. SC2 2017*, vol. 2018-Janua, pp. 112–117, 2018, doi: 10.1109/SC2.2017.24.

[43] Z. Wei-guo, M. Xi-lin, and Z. Jin-zhong, "Research on kubernetes' resource scheduling scheme," *ACM Int. Conf. Proceeding Ser.*, pp. 144–148, 2018, doi: 10.1145/3290480.3290507.

[44] W. Ren, W. Chen, and Y. Cui, "Dynamic Balance Strategy of High Concurrent Web Cluster Based on Docker Container," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 466, no. 1, 2018, doi: 10.1088/1757-899X/466/1/012011.

[45] G. P. P. Geethu and S. K. Vasudevan, "An in-depth analysis and study of Load balancing techniques in the cloud computing environment," *Procedia Comput. Sci.*, vol. 50, pp. 427–432, 2015, doi: 10.1016/j.procs.2015.04.009.