Power-Aware Container Placement Mechanism for CaaS Systems

Abdulelah Alwabel
Department of Computer Sciences
Prince Sattam Bin Abdulaziz University
AlKharj, Saudi Arabia
e-mail: {a.alwabel}@psau.edu.sa

***Abstract*—Containers-as-a-Service (CaaS) platforms are widely adopted due to the lightweight and portable nature of containerized-based applications. However, the growing complexity of container workloads makes it difficult to both achieve efficient resource allocation and reduced energy consumption. To address this challenge, we propose a Power-aware Container Placement (PCP) mechanism, , which maps containers to physical machines (PMs) in cloud data centers with an aim to minimize power consumption while maintaining an acceptable level of performance. The PCP mechanism utilizes the Whale Optimization Algorithm (WOA) to search for energy-aware placements under resource constraints, and then applies an additional best-fit optimization phase to further consolidate and reduce the number of active PMs. The proposed approach is evaluated using a Java-based simulation and is compared against the Extend Directed Container Placement (EDCP) mechanism from the literature. Experimental results show that the proposed mechanism reduces power consumption by approximately 7.4% in comparison to the EDCP mechanism. In addition, it also achieves a about 5% reduction in search time. These results indicate that our mechanism can improve energy efficiency in CaaS systems with minimum negative impact on performance of the systems.**

Keywords-Container; Container Placement; CaaS; PCP; Cloud; Optimization.

I. Introduction

Container Technology is widely employed in Cloud systems as a Container-as-a-Service (CaaS) because they are lightweight and highly portable [1]. However, the rapid growth in the size and complexity of container workloads makes it more challenging to allocate resources effectively while meeting performance requirements and limiting energy consumption. Container placement plays a crucial role in tackling these challenges, as it specifies how containers are assigned to Physical Machines (PMs). Effective placement strategies can improve data-center efficiency by reducing operational costs, lowering power usage, and increasing resource utilization [2].

In real-world deployments, container-based services often consist of many isolated containers, which increases the importance of efficient scheduling and orchestration [3]. As a result, a wide range of placement methods has been proposed, including optimization-based techniques, such as linear programming (e.g., [4]) and heuristic or metaheuristic algorithms (e.g., [5]). Although these approaches can enhance energy efficiency, utilization, or network cost, they may suffer from high computational complexity, added runtime overhead, or limited scalability in large-scale environments. Hybrid optimization approaches are therefore considered a promising alternative, as they can simultaneously balance multiple objectives, such as energy efficiency, load distribution, and quality of service (QoS). This paper introduces a new container placement mechanism that aims to reduce power consumption in CaaS systems while preserving an acceptable performance level compared with existing metaheuristic solutions. The proposed work extends our previous works in [6] and [7] with an aim of further enhancing power efficiency.

The rest of the paper is structured as follows. Section II surveys the related literature. Section III details the proposed approach. Section IV reports the evaluation methodology and discusses the results. Section V concludes the paper.

II. Related Work

The work in [8] presented an energy-aware scheduling model that applies particle swarm optimization. Results showed notable energy savings while preserving an acceptable QoS level.

The authors in [9] proposed an optimization method based on the ant colony optimization algorithm [10] to balance resource utilization, network consumption, and failure events in microservices cloud applications. Their mechanism improved service reliability, load balancing, and network transmission overhead, but it did not account for energy consumption.

A heterogeneous container placement strategy was proposed in [11] to improve the cost efficiency of container orchestration in Kubernetes-based cloud systems. Experiments demonstrated considerable cost reductions compared to default Kubernetes behavior under different workload patterns, although the approach was not evaluated under high service-demand scenarios.

In [12], a locality-aware scheduling model was introduced to boost the performance of containerized cloud services by mitigating resource contention and improving network efficiency. The model employs load-balancing heuristics to enhance application performance; however, performance may degrade sharply as workload size grows [13].

The authors in [14] proposed a scaling mechanism that adapts groups of containers to changing workloads to improve Quality of Experience (QoE). Likewise, [15] focuses on scheduling that prioritizes QoE over QoS by using deep learning to integrate QoE indicators, offering a more faithful representation of user satisfaction for QoE prediction. Their experiments showed an average QoE improvement of about 62% compared with traditional schedulers.

The researchers in [5] developed a whale-optimization-based container placement method [16] to improve resource utilization and reduce power consumption in cloud systems. The

results indicated better performance than competing placement mechanisms in heterogeneous test settings. However, the study did not assess potential performance overheads, such as resource overutilization, associated with the method.

The authors in [17] proposed a multi-objective container migration strategy based on a Binary Grey Wolf Optimizer to improve resource utilization and energy efficiency. To reduce migration inefficiencies, they introduced a node coordination matrix model to address resource fragmentation. The evaluation showed that the proposed strategy outperformed existing mechanisms, suggesting practical effectiveness, yet it did not examine scenarios involving very large containers or large numbers of PMs.

Our earlier work in [6] presented a whale-optimization-based container placement mechanism aimed at reducing energy consumption in cloud systems. However, it adversely affected performance metrics such as service violations and search time. This was extended in [7] by introducing a new scoring scheme to better balance performance with power usage. Although performance improved, some power overhead remained. However, both approaches could be further improved by reducing power consumption while maintaining minimal impact on performance, as will be discussed later.

The authors in [18] introduced a container consolidation approach for CaaS clouds using Fractional Pelican Hawks Optimization. The method includes (i) a host-status module to predict PM load and (ii) a consolidation module to manage container migrations. It targets multiple objectives—energy consumption, resource utilization, and cost—leading to notable efficiency gains. Nonetheless, the study did not evaluate scalability for larger, more complex cloud infrastructures.

Gouaouri et al. [19] study power-aware workload scheduling in Kubernetes, arguing that default rule-based scheduling can lead to higher power consumption and inefficient cluster operation at scale. They propose a two-stage, multi-objective framework that uses an ML-based power predictor trained from real node profiling and then performs online scheduling via TOPSIS and NSGA-II. The solution is implemented as a scalable Kubernetes scheduler plugin that can integrate with other plugins and lets users extend objectives based on QoS needs, explicitly optimizing consolidation (active nodes), load balancing, and utilization efficiency together.

The study proposed by [20] address dynamic container placement for serverless edge computing, where network conditions and limited resources make static placement ineffective and cold-start delays can hurt latency-sensitive tasks. They model placement as an online convex optimization problem and develop a strongly adaptive container placement algorithm with a regret guarantee to minimize average request delay using iterative feedback-based refinement.

The authors in [21] focus on container-based microservice scheduling, noting that existing approaches can suffer from poor load balancing, high network overhead, and weaker reliability. They introduce an enhanced multi-objective PSO that uses an indicator for controlled progress plus complementary indicators to better evaluate trade-offs. The reported results indicate that the proposed mechanism reduces load imbalance and network transmission while improving service reliability in heterogeneous cloud environments.

The authors in [22] propose an SLO-aware container orchestration approach for Kubernetes, motivated by the difficulty of meeting response-time SLOs without costly over-provisioning and by the fact that conventional autoscalers rely on low-level CPU/memory metrics.

Algorithm 1 PCP Mechanism

input: c_l, p_l
initialize s_l as a list of n_s random solutions
for i to $itrN$ **do**
  $b \leftarrow findBestSolution(s_l)$
  foreach s in s_l **do** //update the current solution s
    Let s_r be a random solution
    $cf_1 \leftarrow 2 \times a \times rand$
    $cf_2 \leftarrow 2 \times rand$
    Let $prob$ be a random number from 0 to 1
    if $prob < 0.5$ **then**
      $dir \leftarrow |cf_2 \times s_r - s|$
      if $|cf_1| < 1$ **then**
        $s \leftarrow b - cf_1 \times dir$
      else
        $s \leftarrow s_r - cf_1 \times dir$
      end if
    else
      $dir \leftarrow (|b - s|)$
      Let z be a random number from -1 to 1
      $s \leftarrow dir \times e^z \times cos(2\pi z) \times s$
    end if
  end for
end for
$b \leftarrow optimizeSolution(b)$
return: b

Algorithm 2 Optimization Mechanism

input: b
$containerList \leftarrow b.removeContinaerList()$
$pmList \leftarrow b.removePmList()$
sort $containerList$ by CPU requirement ascending
foreach $container$ in $containerList$ **do**
  foreach pm in $pmList$ **do**
    if $pm.isBestFit()$ **then**
      $pm.host(container)$
      $pmList.add(pm)$
    end if
  end for
end for
$b.setPMList(pmList)$
return: b

III. Proposed Mechanism

In this section, we propose Power-aware Container Placement (PCP) mechanism that places a list of container to a list PMs in a CaaS system with an aim to reduce power consumption. The PCP extends our previous work presented in [7]. The mechanism employs whale optimization algorithm (WOA) that is proposed by [16]. The WOA is a nature-inspired meta-heuristic optimization method that mimics humpback whales' bubble-net hunting behavior. The algorithm begins with a set of random solutions, then in each iteration the search agents update their positions toward either the best solution so far or a random agent, switching between spiral and circular movement to model bubble-net attacking.

The PCP mechanism is depicted in Algorithm 1. It starts by creating a list of n_s number of random solutions, called s_l. Each solution represents placing a container list c_l to a list of PMs found in p_l in a random way. Then, the mechanism finds the best solution, called b, among a list of solutions using a method called $findBestSolution$. The best solution is defined as the solution which places containers to PMs in a manner that improves performance and reduces power consumption as it is explained in [7]. Each solution s_1 is evaluated against another solution s_2 according to the following:

$$ev(s_1, s_2) = \frac{pw(s_1) - pw(s_2)}{pw(s_2)} + \frac{o(s_1) - o(s_2)}{o(s_2)} \quad (1)$$

where $pw(s)$ is the power consumed by the solution s and is calculated as [23]:

$$pw(s) = \sum_{i=1}^{p} (pm_i^{idle} + (pm_i^m - pm_i^{idle}) \times \frac{pm_i^c}{pm_i^t}) \quad (2)$$

where p refers to the total number of PMs in a CaaS system. pm_i^{idle} denotes the power consumption by pm_i when it is idle. pm_i^m refers to the maximum power consumed by this PM. pm_i^c and pm_i^t denote the current CPU and total CPU of pm_i respectively.

$o(s)$ in (1) denotes the number of PMs in a solution where each PM's utilization level exceeds a certain threshold, called ou. If a utilization level in a PM exceeds a certain threshold, this can lead to a negative impact on the performance of this PM [24].

b is, then, used to update other solutions based on a method called $updateSolution$ according to [5]. The same process is repeated for a number of iterations called $itrN$ as it is explained in [6]. The PCP mechanism repeats this process of updating solutions in s_l until $itrN$.

The last step is to further optimize b one last time using an optimization approach as it is depicted in Algorithm 2. The optimization mechanism employs best fit (BF) approach in order to reduce the number of active PMs in the system. It sorts containers by CPU requirements in an ascending manner. Then, a single PM will host as many containers as it can handle in terms of CPU and memory powers. Lastly, the updated best solution is returned as a result of this optimization phase.

TABLE I. Experiment Configurations

Parameter	Values
Container Specifications:	
Number	2000
CPU Requirements (MIPS)	256, 512, 1024, 2048, 4096
Memory Requirements (MB)	128, 256, 512, 1024, 2048
PM Specifications:	
No of PMs	1000
No of Cores	6, 8, 20, 32, 56, 64
CPU (GHz)	2.2, 2.7, 2.9, 3.2
Memory (GB)	16, 128, 192, 512, 768
pm_{idle} (watt)	15.6, 21.7, 48.6, 53.2, 127
pm_m (watt)	58.9, 82.8, 269, 291, 377, 410
Simulation Settings:	
n_s	5
$itrN$	10
ou	80%

IV. Evaluation

This section discusses the results of PCP mechanism and compares it with EDCP mechanism [7]. The EDCP mechanism is a container placement mechanism based on Meta-heuristic optimization algorithm that places containers with an aim to find an optimum solution in an acceptable time. The mechanisms are evaluated in terms of performance and energy consumption.

A. Experiment Configurations

Table I shows the configurations of the experiments conducted in a Java simulation that was extended based on a work by [6] to evaluate the EDCP and PCP mechanisms. The total number of container is 2000 that submitted to a CaaS of PM machines. The specifications of the PMs are listed in the table. The table shows power consumption of each PM when it is idle and when it is at maximum loading. ou is set to 80% because several studies show that when CPU utilization exceeds this level, the system may experience performance degradation [25].

B. Results

The PCP mechanism outperformed the EDCP mechanism in reducing power consumption as it is depicted in Figure 1. The Figure clearly demonstrates that the gap in power consumption increases when the number of container increases. Overall, the proposed mechanism reduces the power by about 7.4% in compared to the EDCP mechanism.

With respect to search time, PCP also achieves an improvement in by reducing the search time to find an optimum solution by about 5% on average. The EDCP scored about 0.27

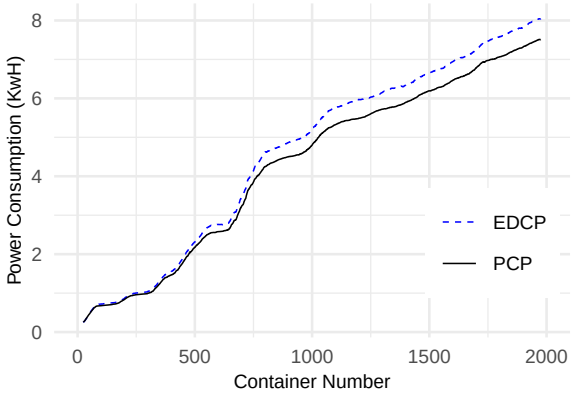

Figure 1. Power Consumption

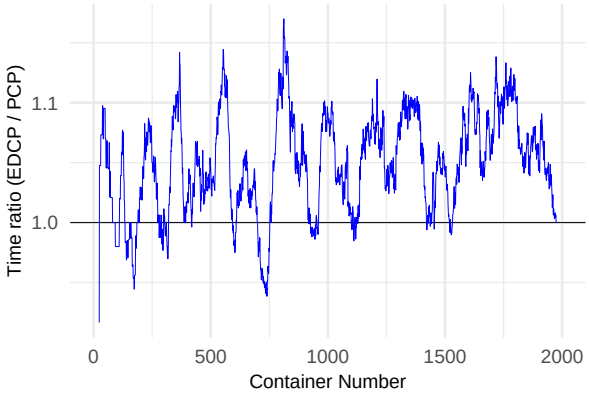

Figure 2. Search Time

seconds on average to find the best solution while the PCP achieved about 0.26 seconds on average. The t-test analysis demonstrates that analysis the PCP achieved a significantly lower runtime than the EDCP (paired t-test: t(2000)=7.63, $p < 0.001$). Figure 2 demonstrates the time ratio between EDCP and PCP which is inconsistent. That means although the PCP mechanism managed to reduce time in general there are some instances that the EDCP mechanism behaved better.

V. Conclusion and Future Work

Container technologies are widely adopted in CaaS environments due to their lightweight virtualization and portability. Container placement can play a key role in tackling several challenges in CaaS including effective resource allocation and limiting energy consumption. Effective container placement solutions can improve efficiency in clouds' infrastructure by reducing operational costs, lowering power usage, and increasing resource utilization.

In this paper, we presented PCP, a novel container placement mechanism that maps containers to PMs in cloud data centers with the objective of minimizing energy consumption. The proposed approach adapts the WOA to search for an energy-aware placement that satisfies resource constraints. Moreover, PCP refines an existing optimization technique from the literature to further reduce power consumption. Experimental results demonstrate that PCP achieves approximately a 7.4% reduction in power consumption compared with EDCP.

For future work, the proposed mechanism may be extended from static placement to dynamic runtime management by supporting container or VM migrations among PMs whenever workload fluctuations or resource imbalance occur. Moreover, the heterogeneity of physical machines can be incorporated more explicitly into the placement logic by accounting for variations in CPU capacity, memory resources, and power profiles. The current evaluation also relies on a simplified linear power model based on CPU utilization; therefore, it would be valuable to investigate whether the effectiveness of the proposed approach is preserved under more sophisticated non-linear or empirical power models. Furthermore, stronger performance guarantees for hosted containers could be introduced by integrating QoS/SLO-aware objectives and constraints, such as latency, response time, and service stability, instead of relying primarily on CPU-utilization threshold control. In addition, PCP can be extended with reliability-aware capabilities to improve service continuity in CaaS environments. Large-scale CaaS infrastructures typically operate a substantial number of PMs, which increases the likelihood of host failures and may consequently degrade SLOs. Incorporating fault-tolerance mechanisms (e.g., replication, proactive migration, or failure-aware placement constraints) could mitigate these disruptions and enhance overall system reliability.

References

[1] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes", *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, Sep. 2014, ISSN: 2325-6095. DOI: 10.1109/MCC.2014.51.

[2] I. Ahmad, M. G. AlFailakawi, A. AlMutawa, and L. Alsalman, "Container scheduling techniques: A Survey and assessment", *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 7, pp. 3934–3947, Jul. 2022, ISSN: 13191578. DOI: 10.1016/j.jksuci.2021.03.002.

[3] N. Zhou, H. Zhou, and D. Hoppe, "Containerization for High Performance Computing Systems: Survey and Prospects", *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2722–2740, Apr. 2023, ISSN: 0098-5589. DOI: 10.1109/TSE.2022.3229221. arXiv: 2212.08717.

[4] A. Mseddi, W. Jaafar, H. Elbiaze, and W. Ajib, "Joint Container Placement and Task Provisioning in Dynamic Fog Computing", *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 028–10 040, 2019, ISSN: 23274662. DOI: 10.1109/JIOT.2019.2935056.

[5] A. Al-Moalmi, J. Luo, A. Salah, K. Li, and L. Yin, "A whale optimization system for energy-efficient container placement in data centers", *Expert Systems with Applications*, vol. 164, p. 113 719, Feb. 2021, ISSN: 09574174. DOI: 10.1016/j.eswa.2020.113719.

[6] A. Alwabel, "A Novel Container Placement Mechanism Based on Whale Optimization Algorithm for CaaS Clouds", *Electronics*, vol. 12, no. 15, p. 3369, Aug. 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12153369.

[7] A. Alwabel, "An extended container placement mechanism to enhance the efficiency of cloud systems", *PeerJ Computer Science*, vol. 11, e3348, Nov. 2025, ISSN: 2376-5992. DOI: 10.7717/peerj-cs.3348.

[8] T. Shi, H. Ma, and G. Chen, "Energy-Aware Container Consolidation Based on PSO in Cloud Data Centers", in *2018 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, Jul. 2018, pp. 1–8, ISBN: 978-1-5090-6017-7. DOI: 10.1109/CEC.2018.8477708.

[9] M. Lin, J. Xi, W. Bai, and J. Wu, "Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud", *IEEE Access*, vol. 7, pp. 83 088–83 100, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2924414.

[10] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization", *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, Nov. 2006, ISSN: 1556-603X. DOI: 10.1109/MCI.2006.329691.

[11] Z. Zhong and R. Buyya, "A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources", *ACM Transactions on Internet Technology*, vol. 20, no. 2, pp. 1–24, May 2020, ISSN: 1533-5399. DOI: 10.1145/3378447.

[12] D. Zhao, M. Mohamed, and H. Ludwig, "Locality-Aware Scheduling for Containers in Cloud Computing", *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 635–646, Apr. 2020, ISSN: 2168-7161. DOI: 10.1109/TCC.2018.2794344.

[13] S. Deng et al., "Cloud-Native Computing: A Survey From the Perspective of Services", *Proceedings of the IEEE*, vol. 112, no. 1, pp. 12–46, Jan. 2024, ISSN: 0018-9219. DOI: 10.1109/JPROC.2024.3353855.

[14] G. Santos, H. Paulino, and T. Vardasca, "QoE-aware autoscaling of heterogeneous containerized services (and its application to health services)", in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, New York, NY, USA: ACM, Mar. 2020, pp. 242–249, ISBN: 9781450368667. DOI: 10.1145/3341105.3373915.

[15] M. Carvalho and D. F. Macedo, "Container Scheduling in Co-Located Environments Using QoE Awareness", *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3247–3260, Sep. 2023, ISSN: 1932-4537. DOI: 10.1109/TNSM.2023.3244090.

[16] S. Mirjalili and A. Lewis, "The Whale Optimization Algorithm", *Advances in Engineering Software*, vol. 95, pp. 51–67, May 2016, ISSN: 09659978. DOI: 10.1016/j.advengsoft.2016.01.008.

[17] C. L. Chuqiao Lin et al., "Container Migration Strategy Based on Multi-objective Optimization for Edge-Cloud Coordination enabled Smart Grids", *Journal of Computers*, vol. 34, no. 6, pp. 047–062, Dec. 2023, ISSN: 19911599. DOI: 10.53106/199115992023123406004.

[18] M. K. Patra, B. Sahoo, and A. K. Turuk, "FPHO: Fractional Pelican Hawks optimization based container consolidation in CaaS cloud", *Concurrency and Computation: Practice and Experience*, vol. 36, no. 12, e8052, May 2024, ISSN: 1532-0626. DOI: 10.1002/cpe.8052.

[19] M. D. E. Gouaouri, S. Ouahouah, M. Bagaa, M. A. Ouameur, and A. Ksentini, "A multi-objective framework for power-aware scheduling in kubernetes", *IEEE Transactions on Network and Service Management*, vol. 23, pp. 28–46, 2026, ISSN: 1932-4537. DOI: 10.1109/TNSM.2025.3630045.

[20] S. Cao and S. Wang, "Dynamic container placement at serverless edge: An online convex optimization method", in *2025 IEEE 34th Wireless and Optical Communications Conference (WOCC)*, IEEE, May 2025, pp. 66–70, ISBN: 979-8-3315-3928-3. DOI: 10.1109/WOCC63563.2025.11082211.

[21] M. Douiri, I. B. Mansour, and M. Tagina, "Optimizing container-based microservice scheduling with parallel particle swarm based on diversity indicators", in *2025 IEEE 37th International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, Nov. 2025, pp. 81–88, ISBN: 979-8-3315-4919-0. DOI: 10.1109/ICTAI66417.2025.00019.

[22] A. Marchese and O. Tomarchio, "Slo-aware container orchestration on kubernetes clusters", in *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*, IEEE, Jul. 2025, pp. 318–327, ISBN: 979-8-3315-5557-3. DOI: 10.1109/CLOUD67622.2025.00040.

[23] A. A. Khan et al., "An energy and performance aware consolidation technique for containerized datacenters", *IEEE Transactions on Cloud Computing*, 2019, ISSN: 2168-7161. DOI: 10.1109/tcc.2019.2920914.

[24] I. Çağlar and D. T. Altılar, "Look-ahead energy efficient VM allocation approach for data centers", *Journal of Cloud Computing*, vol. 11, no. 1, p. 11, Dec. 2022, ISSN: 2192-113X. DOI: 10.1186/s13677-022-00281-x.

[25] B. Gregg, *Systems Performance: Enterprise and the Cloud*, 1st. Pearson Education, Inc., 2014.