PeerJ Computer Science

An extended container placement mechanism to enhance the efficiency of cloud systems

Abdulelah Alwabel

Department of Computer Sciences, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj, Saudi Arabia

ABSTRACT

The rapid increase of containerized applications in cloud environments has highlighted the critical need for efficient resource management and energy optimization. This article extends our previous work with an aim to enhance the performance of cloud systems. We propose an Extended Directed Container Placement (E_DCP) mechanism, a novel approach designed to enhance container placement efficiency in cloud systems when the number of container increases significantly. Leveraging the Whale Optimization Algorithm (WOA), the mechanism utilizes a scoring mechanism to evaluate various solutions with an aim to identify the best among them. By optimizing multiple objectives, including search time, resource utilization and energy efficiency, the mechanism achieves superior outcomes in heterogeneous and homogeneous cloud infrastructures in comparison to recent methods. The mechanism optimizes this solution to minimize overutilized physical machines. Extensive simulations demonstrate significant improvements in search time and resource utilization with acceptable energy consumption level.

Subjects Algorithms and Analysis of Algorithms, Computer Networks and Communications, Distributed and Parallel Computing, Mobile and Ubiquitous Computing, Optimization Theory and Computation

Keywords Container placement, Cloud, CaaS, E_DCP

Submitted 3 February 2025
Accepted 9 October 2025
Published 4 November 2025

Corresponding author
Abdulelah Alwabel,
a.alwabel@psau.edu.sa

Academic editor
Elad Michael Schiller

Additional Information and Declarations can be found on page 20

DOI **10.7717/peerj-cs.3348**

INTRODUCTION

The swift advancement of cloud computing has revolutionized resource management and application deployment, enabling dynamic scalability and enhanced cost efficiency (*Buyya, Yeo & Venugopal, 2008*). Recently, a wide range of cloud service providers utilize containerization technology (*Pahl, 2015*) instead of virtualization technology (*Rosenblum & Garfinkel, 2005*) to host a wide range of cloud applications (*Alahmad, Daradkeh & Agarwal, 2018*). Lightweight isolation platforms, such as Docker (*Merkel, 2014*) and LXC (*Randal, 2020*), have gained widespread adoption as they enable virtualization without the need to boot an entire virtual machine. Containers can typically launch within a few seconds leveraging a user-level engine (*Ahmed et al., 2019*).

Within Containers-as-a-Service (CaaS) environments, containerized applications have emerged as a popular choice due to their lightweight and portable nature (*Bernstein, 2014*). Containerized applications have emerged as a cornerstone of modern cloud and edge computing architectures due to their lightweight, portable, and efficient design. Unlike traditional virtualization styles that rely on full operating system virtualization, containers

enable isolated execution environments while sharing the host kernel, resulting in reduced overhead and faster launch times (*Pahl et al., 2019*). This efficiency makes them highly suitable for microservices-based systems, which demand rapid scalability and dynamic resource management (*Zhou, Zhou & Hoppe, 2023*). Furthermore, the widespread adoption of container orchestration frameworks has significantly enhanced the deployment and management of containerized workloads, promoting resilience and high availability in distributed environments.

However, as containerized workloads grow in complexity and scale, ensuring efficient resource allocation and performance improvements while minimizing energy consumption has become a critical challenge. Effective container placement plays a vital role in dealing with these concerns by optimizing the placement of containers to physical machines (PMs). Container placement mechanisms can contribute in performance enhancement in various aspects in data centers such as minimizing operational costs, reducing energy consumption and/or improving resource utilization (*Ahmad et al., 2022*).

Containerized based applications can become complex as large number of isolated containers may be required in production which necessitates efficiency in container scheduling and management (*Zhou, Zhou & Hoppe, 2023*). Various strategies for container placement have been proposed in recent years, ranging from linear programming models such as the work by *Mseddi et al. (2019)* to heuristic and metaheuristic algorithms such as the work by *Al-Moalmi et al. (2021)*. While these methods have demonstrated improvements in energy efficiency, resource utilization, or network cost reduction, they often suffer from limitations such as high computational complexity, performance overhead, or scalability issues in large-scale infrastructures. Hybrid optimization approaches offer a promising avenue to overcome these challenges by balancing multiple objectives such as energy efficiency, load balancing, and quality of service (QoS).

In this study, we introduce an Extended Directed Container Placement (E_DCP) mechanism, a novel container placement that extends our prior research (*Alwabel, 2023*) and introduces a scoring-based evaluation strategy leveraging the Whale Optimization Algorithm (WOA) (*Mirjalili & Lewis, 2016*). The contribution of this article is as follows. Firstly, this work proposes a scoring mechanism to evaluate various solutions with an aim to identify the best among them. By optimizing multiple objectives, including search time, resource utilization and energy efficiency.

To assess resource utilization, this article introduces the overutilized level threshold (OLT) metric within its scoring mechanism. The OLT defines a specific utilization level for a computer machine; if this threshold is exceeded, the machine's performance can degrade, compromising overall system performance and service quality (*Katal, Dahiya & Choudhury, 2023*). Furthermore, overutilized machines are more susceptible to hardware failures due to overheating, which increases the frequency of system outages and raises maintenance costs (*Lee, Viswanathan & Pompili, 2017*).

Secondly, designing a multi-objective optimization framework to enhance container placement efficiency in both homogeneous and heterogeneous cloud environments. The

mechanism develops an optimization method which reduces the number of overutilized PMs to improve performance. Thirdly, The proposed mechanism enhances container placement by addressing critical limitations of previous approaches, scalability, search time inefficiencies, and utilization when the number of containers grows.

The mechanism outperforms recent methods in both heterogeneous and homogeneous cloud infrastructures in terms of search time and resource utilization, particularly as the number of containers increases significantly. Extensive simulations demonstrate significant improvements in search time and resource utilization with acceptable energy consumption level. This research contributes to advancing the performance of containerized cloud systems, offering practical insights for large-scale deployments.

The reminder of this article is organized as follows: 'Related Works' section discusses the state-of-art works in container placement area. 'Proposed Mechanism' proposes our mechanism. The findings of this article is presented in 'Experiments and Evaluation'. The conclusion of this article is drawn in 'Conclusion'.

RELATED WORKS

The authors in *Piraghaj et al. (2015)* addressed a power optimization problem in the CaaS environment. The proposed mechanism effectively improved energy efficiency by optimizing container placement and reducing the number of active servers. They introduced a correlation-aware placement algorithm, demonstrating that overload and underload threshold algorithms performed better than others, particularly when migrating larger containers.

A dynamic resource placement mechanism in heterogeneous CaaS data centers was proposed by *Mao et al. (2017)*. The mechanism selects the PM to deploy a container based on both the available resources and service demand. Performance comparison with similar approaches demonstrated a more efficient and balanced usage of resources. However, network consumption was high as a result of employing this mechanism (*Carvalho & Macedo, 2023*).

The authors in *Zhang et al. (2017)* introduced a container placement based on linear programming model (ILP) for allocating containers to computing nodes, taking into account various optimization criteria, including energy consumption and network cost. Compared to Docker Swarm's binpack strategy, the placement mechanism achieved approximately a 45% cost reduction. Another ILP based approach was developed by *Wan et al. (2018)* with an aim to minimize cost. The approach accounted for networking costs and addressed the scheduling problem in a distributed manner. However, the approach suffered of high computational complexity and overlooked key optimization factors, such as energy consumption and response time (*Ahmad et al., 2022*).

The authors in *Shi, Ma & Chen (2018)* proposed an energy-based model for optimizing container scheduling that is based on a particle swarm optimization approach. The results demonstrated that the proposed method has shown significant improvements in energy efficiency while maintaining an acceptable level of QoS.

The authors in *Lin et al. (2019)* developed an optimization technique that is based on ant colony optimization algorithm (*Dorigo, Birattari & Stutzle, 2006*) with an aim to trade off between resource utilization, network consumption, and failure events of microservices cloud applications. The proposed mechanism demonstrated efficient outcome in service reliability, load balancing and network transmission overhead. The mechanism, however, failed to consider energy consumption.

A novel heterogeneous container placement strategy is presented in *Zhong & Buyya (2020)* to enhance the cost efficiency of container orchestration in Kubernetes-based cloud systems. The experiments demonstrated significant cost savings compared to the default Kubernetes framework under various workload patterns. However, their work was not tested under conditions of high service demand.

A article presents a novel locality-aware scheduling model aimed at improving the performance of containerized cloud services by addressing key challenges in resource contention and network efficiency (*Zhao, Mohamed & Ludwig, 2020*). The model implements load balance heuristic to improve application performance. The performance, however, the proposed mechanism could dramatically degrade when the workloads scales up (*Deng et al., 2024*).

The authors in *Santos, Paulino & Vardasca (2020)* proposed a novel scaling mechanism to scale a group of containers to various workloads with an aim to improve quality of experience (QoE) metrics. Similarly, the work presented in *Carvalho & Macedo (2023)* emphasizes improving service scheduling by prioritizing QoE over QoS objectives. It utilizes deep learning models to incorporate QoE metrics, providing a more accurate representation of user satisfaction to predict user QoE. Experimental results demonstrated that the proposed scheduler improved average QoE by about 61.5% compared to conventional schedulers.

The researchers in *Al-Moalmi et al. (2021)*, developed a container placement mechanism based on whale optimization algorithm (*Mirjalili & Lewis, 2016*) in order to optimize resource utilization and power consumption in cloud systems. The results showed that the placement mechanism is superior over other placement mechanism in heterogeneous test environment. However, the study did not take the performance overhead such as resource over utilization of this method into consideration.

The authors in *Chuqiao Lin et al. (2023)* designed a novel multi-objective container migration strategy based binary grey wolf optimizer algorithm on with an aim to optimize resource utilization and energy consumption. To enhance the efficiency of container migration, it established a node coordination matrix model to address resource fragmentation. The results of this study demonstrate that the proposed strategy outperformed existing mechanism, indicating its effectiveness in real-world applications. However, the study did not address the issue of large container and PM number in the evaluation.

Our previous work in *Alwabel (2023)* presented a novel container mechanism that utilizes a meta-heuristic algorithm called whales optimization algorithm with an aim to minimize energy consumption cloud systems. However, this work has a negative impact on performance criteria such as service violation and search time.

Table 1 Comparison of related works and our work.

Work	Power-aware	QoS/QoE	Cost efficiency	Resource utilization	Heavy demand
Piraghaj et al. (2015)	✓	✗	✗	✗	✗
Mao et al. (2017)	✗	✗	✓	✗	✗
Zhang et al. (2017)	✗	✗	✓	✗	✗
Wan et al. (2018)	✗	✓	✗	✗	✗
Shi, Ma & Chen (2018)	✓	✓	✗	✗	✗
Lin et al. (2019)	✗	✓	✗	✓	✗
Zhong & Buyya (2020)	✗	✗	✓	✗	✗
Zhao, Mohamed & Ludwig (2020)	✗	✗	✗	✓	✗
Santos, Paulino & Vardasca (2020)	✗	✓	✗	✗	✗
Carvalho & Macedo (2023)	✗	✓	✗	✗	✗
Al-Moalmi et al. (2021)	✓	✗	✗	✓	✗
Chuqiao Lin et al. (2023)	✓	✗	✗	✓	✗
Alwabel (2023)	✓	✗	✓	✓	✓
Patra, Sahoo & Turuk (2024)	✓	✗	✓	✓	✗
Our work	✓	✓	✓	✓	✓

The authors in *Patra, Sahoo & Turuk (2024)* proposed a method for container consolidation in CaaS cloud systems using fractional pelican hawks optimization. The proposed approach includes two main modules: a host status module for predicting PM load and a consolidation module for managing container migration. The method aims to optimize multiple objectives, including energy consumption, resource utilization, and costs, to achieving significant efficiency improvements. However, the scalability of the proposed method in larger, more complex cloud infrastructures was not addressed.

In conclusion, this section discusses several related works and their focus o the area of task scheduling mechanisms. Table 1 presents a comparison of related works with our work in terms of power consumption, Qos/QoE, cost reduction, resource utilization and heavy demand. The next section explains our work in more details.

PROPOSED MECHANISM

While prior methods have achieved significant improvements in energy efficiency and resource utilization, they often neglect the interplay between search time and performance degradation in the presence of large number of tasks. The E_DCP mechanism addresses these gaps through an optimized approach, balancing multiple objectives in both homogeneous and heterogeneous environments. This mechanism extended our previous work proposed in *Alwabel (2023)*. Figure 1 depicts the E_DCP mechanism. The first phase of this algorithm is to create a *NW* of solutions. Each solution is initialized by placing a list of containers randomly to a number of PMs. Algorithm 1 illustrates this allocation process. This process is inspired by the concept of WOA approach.

The mechanism identifies the best solution among this list of solution using the Algorithm 2. The algorithm employs an evaluation mechanism *EV* in order to compare two solutions s_1 and s_2 to determine which one yields a better outcome using a scoring

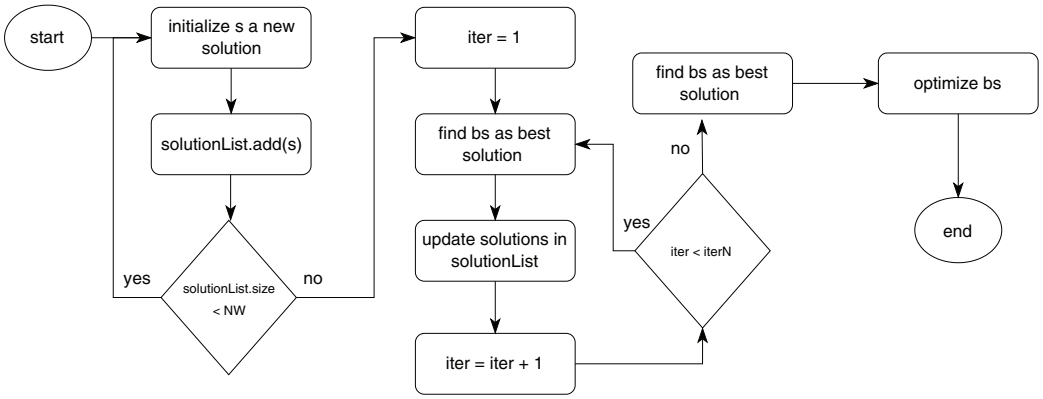

Figure 1 **The proposed mechanism.** Full-size DOI: 10.7717/peerj-cs.3348/fig-1

Algorithm 1 **Solution initialization.**

```
1: input: containerList, PMList
2: foreach container in containerList do
3:   pm ← pmList.getRandom()
4:   if pm.canHost(container) then
5:     pm.add(container)
6:   else
7:     go to line 3
8:   end if
9: end for
```

Algorithm 2 **Find best solution.**

```
1: input: solutionList
2: bs ← solutionList.get(0)
3: for each cs in solutionList do
4:   if evaluteSolution(cs, bs) <0 then
5:     bs = cr
6:   end if
7: end for
```

mechanism. The scoring mechanism is based on three evaluation criteria: energy consumption *powe*, utilization level *olt* and number of active PMs *apm*. The evaluation is given as follows:

$$Ev(s_1, s_2) = pow_{ev}(s_1, s_2) + olt_{ev}(s_1, s_2) + apm_{ev}(s_1, s_2) \tag{1}$$

where $pow_{ev}(s_1, s_2)$ refers to the evaluation of these two solutions in terms of energy consumption. It is given as follows:

$$pow_{ev}(s_1, s_2) = \frac{pow(s_1) - pow(s_2)}{pow(s_2)} \tag{2}$$

where $pow(s_1)$ is the total power consumption for a single solution. Energy efficiency plays a key role in reducing running costs in data centers as well as reducing carbon footprint

(*Gill & Buyya, 2019*). It is calculated according to the authors in *Khan et al. (2019)* as follows:

$$pow(solution) = \sum_{i=1}^{p} (pm_i^{p.idle} + (pm_i^{p.max} - pm_i^{p.idle}) \times U(pm_i)) \tag{3}$$

where p refers to the total number of PMs in a data center. $pm_i^{p.idle}$ denotes the power consumption by a PM i when it remains idle. $pm_i^{p.max}$ refers to the maximum power consumed by this PM. $U(pm_i)$ denotes the utilization level of this machine and is given as follows (*Alwabel, 2023*):

$$U(pm_i) = \frac{pm_i^{cur.cpu}}{pm_i^{tot.cpu}} \tag{4}$$

where $pm_i^{cur.cpu}$ and $pm_i^{tot.cpu}$ refers to used CPU and total CPU of PM i respectively. However, the result of using an approach to reduce power consumption can lead to resources being overutilizaed which leads to performance downgrade (*Katal, Dahiya & Choudhury, 2023*). Overutilization of PMs can significantly degrade overall system performance and service quality. When a PM exceeds a certain threshold resource capacity, it may lead to an increase in queuing delays, higher response times, and reduced throughput. Furthermore, overutilized PMs are more prone to hardware failures as a result of overheating, which can increase the frequency of outages and maintenance costs.

As a consequence, overutilized PMs can negatively affect the performance of hosted VMs or containers, potentially violating QoS requirements and SLAs (*Gao et al., 2013*). In dynamic and time-sensitive applications such performance degradation can be particularly critical. Therefore, effective resource scheduling mechanisms are essential to reduce the impact of overutilization and to ensure efficient container placement mechanism in virtualized environments. Let olt_{ev} denote the number of PMs in a solution where each PM's utilization level exceeds a certain threshold. It is calculated as:

$$olt_{ev}(s_1, s_2) = \frac{olt(s_1) - olt(s_2)}{olt(s_2)}. \tag{5}$$

If a utilization level in a server exceeds a certain threshold, this can lead to a negative impact on the performance of this server (*Çağlar & Altılar, 2022*). Therefore, we consider the number of OLT PMs in a solution as an evaluation criteria. It is given as follows:

$$olt(solution) = \sum_{i=1}^{p} oltPM \tag{6}$$

where $oltPM$ refers to over level threshold of PM's utilization level and it is given as follows:

$$oltPM = \begin{cases} 0 & U(pm) < ot \\ 1 & otherwise \end{cases} \tag{7}$$

where ot is an overloaded threshold. So if a PM exceeds this threshold, it is considered overloaded. Improving utilization can lead to reduction in energy consumption when idle

PMs are set to power mode saving (*Manvi & Krishna Shyam, 2014*). However, the downgrade of this approach is that it may cause performance degradation as explained before. The number of active PM (apm) is introduced in order to balance these two conflicting factors. The following equation is given to calculate the number of active PMs of two solutions s_1 and s_2:

$$apm_{ev}(s_1, s_2) = \frac{apm(s_1) - apm(s_2)}{apm(s_2)} \tag{8}$$

where $apm(solution)$ is:

$$apm(solution) = \begin{cases} 0 & U(pm) = 0 \\ 1 & otherwise \end{cases} \tag{9}$$

The next phase is to update these solutions according to the work proposed by *Mirjalili & Lewis (2016)* and explained thoroughly by our previous work in *Alwabel (2023)*. Algorithm 3 details the process of updating solutions. The, best solution is reevaluated against these updated solutions. If there is a solution which yields a better outcome according to Eq. (1), then it becomes the new best solution. This step is repeated for a number of times.

The last step is further optimize the best best solution. This step assumes that the best solution produced by the previous step can have some overloaded PMs. Therefore, the E_DCP utilizes Algorithm 4 to migrate some containers from those PMs to PMs with the lowest utilization level.

EXPERIMENTS AND EVALUATION

This section presents the results of the proposed container placement approach and compares them with state-of-the-art mechanisms. The proposed mechanism is compared with the DCP mechanism (*Alwabel, 2023*) and the IGA mechanism (*Zhang et al., 2019*). The DCP mechanism is a container placement mechanism based on WOA algorithm that places containers with an aim to reduce energy consumption. Similarly, the IGA mechanism is a genetic based placement mechanism that places containers on cloud data centers to reduce power consumption. The mechanisms are evaluated under three criteria: search time, OLT ad energy consumption.

Configuration

In order to comprehensively investigate the effectiveness of the E_DCP mechanism, the number of containers in each test run starts with 500 containers. The number of containers is increased by 500 until it reaches 5,000 containers. Each container has two specifications which are CPU and RAM parameters. These container's specifications are classifies into four types of containers depending on the values of CPU and RAM as it is demonstrated in Table 2.

The mechanisms are evaluated under two infrastructure types: heterogeneous and homogeneous data centers. The heterogeneous data center consists of six different PMs which are listed in Table 3. The table demonstrates the specifications of these PMs and

Algorithm 3 **Solution update.**

input: *cr, bs*
$rs \leftarrow initializeSolution$
foreach *container* in *cr.getContList* **do**
  $crPM_i \leftarrow cr.getPMid(container)$
  $bsPM_i \leftarrow bs.getPMid(container)$
  $rsPM_i \leftarrow rs.getPMid(container)$
  $rand = getRandomNumber()$
  $a \leftarrow rand \times 2$
  $cf_1 \leftarrow 2 \times a \times rand - a$
  $cf_2 \leftarrow 2 \times rand$
  $prob \leftarrow getRandomNumber(0, 1)$
  $dir \leftarrow cf_2 \times rsPM_i - crPM_i$
  if $prob < .5$ and $cf_1 < 1$ **then**
    $tarPM_i = bsPM_i - cf_1 \times dir$
  else if $prob < .5$ and $cf_1 \geq 1$
    $tarPM_i = rsPM_i - cf_1 \times dir$
  else
    $z \leftarrow getRandomNumber(-1, 1)$
    $dir_d \leftarrow bsPM_i - crPM_i$
    $tarPM_i = dir_d \times e^{b \times z} \times cos(2\pi \times z) + bsPM_i$
  end if
  $cr.migrate(container, tarPM_i)$
end for

Algorithm 4 **Solution optimization.**

input: *solution, OLT*
containerList solution.getContinaerList()
pmList solution.getPmList()
foreach *container* in *containerList* **do**
  foreach *pm* in *pmList* **do**
    pm getMinOLTGap(pmList, OLT)
    pmList.update(pm)
  end for
end for
foreach *pm* in *pmList* **do**
  if $pm.getUtilization = 0$ **then**
    pmList.remove(pm)
  end if
end for
solution.setPMList(pmList)

Table 2 **Container types.**

Container type	MIPS (MHz)	RAM (MB)
Small	256, 512	128, 256
Medium	256, 512, 1,024	512, 1,024
Large	1,024, 2,048, 4,096	1,024, 2,048
Multi	Small, Medium, Large	Small, Medium, Large

Table 3 PM types.

PM ID	No of cores	CPU (GHz)	RAM (GB)	p_{idle} (watt)	p_{max} (watt)
1	6	3.2	16	15.6	58.9
2	8	3.2	16	21.7	82.8
3	20	2.2	768	127	291
4	32	2.9	512	96.6	377
5	56	2.7	192	48.6	410
6	64	2.2	128	53.2	269

power parameters as it was listed in *Alwabel (2023)*. For homogeneous environment, the infrastructure has only one PM type which is PM ID 1 in Table 3. Each run tests each container type with ten different container numbers as mentioned earlier. Therefore, the total number of test runs is 80 runs.

The experiments are performed a simulated environment implemented in the Java programming language on a computer machine with a 2.8 GHz Intel Core i7 processor and 16 GB of RAM with a MacOS Ventura OS.

Search time

This section presents our investigation about the search time required to find a solution among the evaluated mechanisms. The search time refers to the time that is required to find a solution (*i.e.* find a hosting PMs to a group of containers). A good mechanism aims at reducing this time in order to improve the efficiency of cloud systems (*Ahmad et al., 2022*).

Figure 2 demonstrates that the proposed mechanism outperformed the DCP and IGA mechanisms in terms of search time in homogeneous infrastructure. For example, the search time to in Fig. 2A that was carried out by E_DCP is 3.1 s while it is 54.94 and 7.06 s for the DCP and IGA mechanisms respectively. Figure 2D shows that Our mechanism reduced the search time by about 94% compared to the DCP mechanism.

The E_DCP mechanism behaved similarly in heterogeneous cloud systems in term of search time as it is depicted in Fig. 3. It outperformed other mechanisms in all container types: small, medium, large and multi as it is demonstrated in Figs. 3A, 3B, 3C and 3D respectively. When the container number is 5,000 and the container type is large, the proposed mechanism's search time is about 3 s while the DCP mechanism scored about 65 s and the IGA scored about 522 s.

OLT

OLT can play an important role in cloud systems as it reflects the performance of such systems. In this experiment, OLT is measured by two factors: the number of PMs that exceed a utilization threshold because if a machine's current utilization exceeds a certain threshold this can lead to performance degradation which can affect the SLA of cloud services (*Beloglazov & Buyya, 2012*). The second factor is the total number of active PMs. The more nodes working at the same time, the more power consumed by a cloud system.

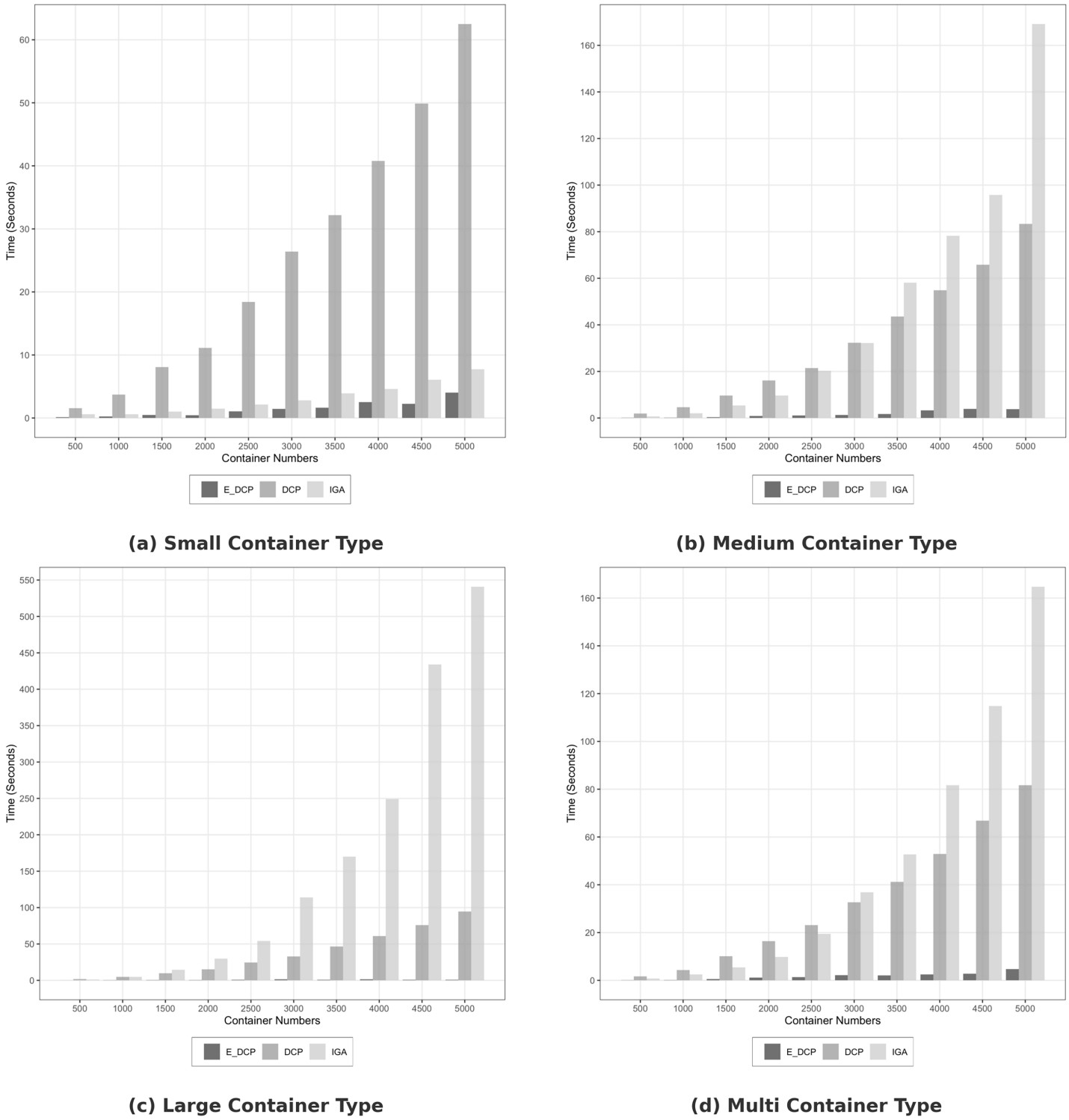

Figure 2 Search time in homogeneous data center. (A) Small container type; (B) Medium container type; (C) Large container type; (D) Multi container type.
Full-size DOI: 10.7717/peerj-cs.3348/fig-2

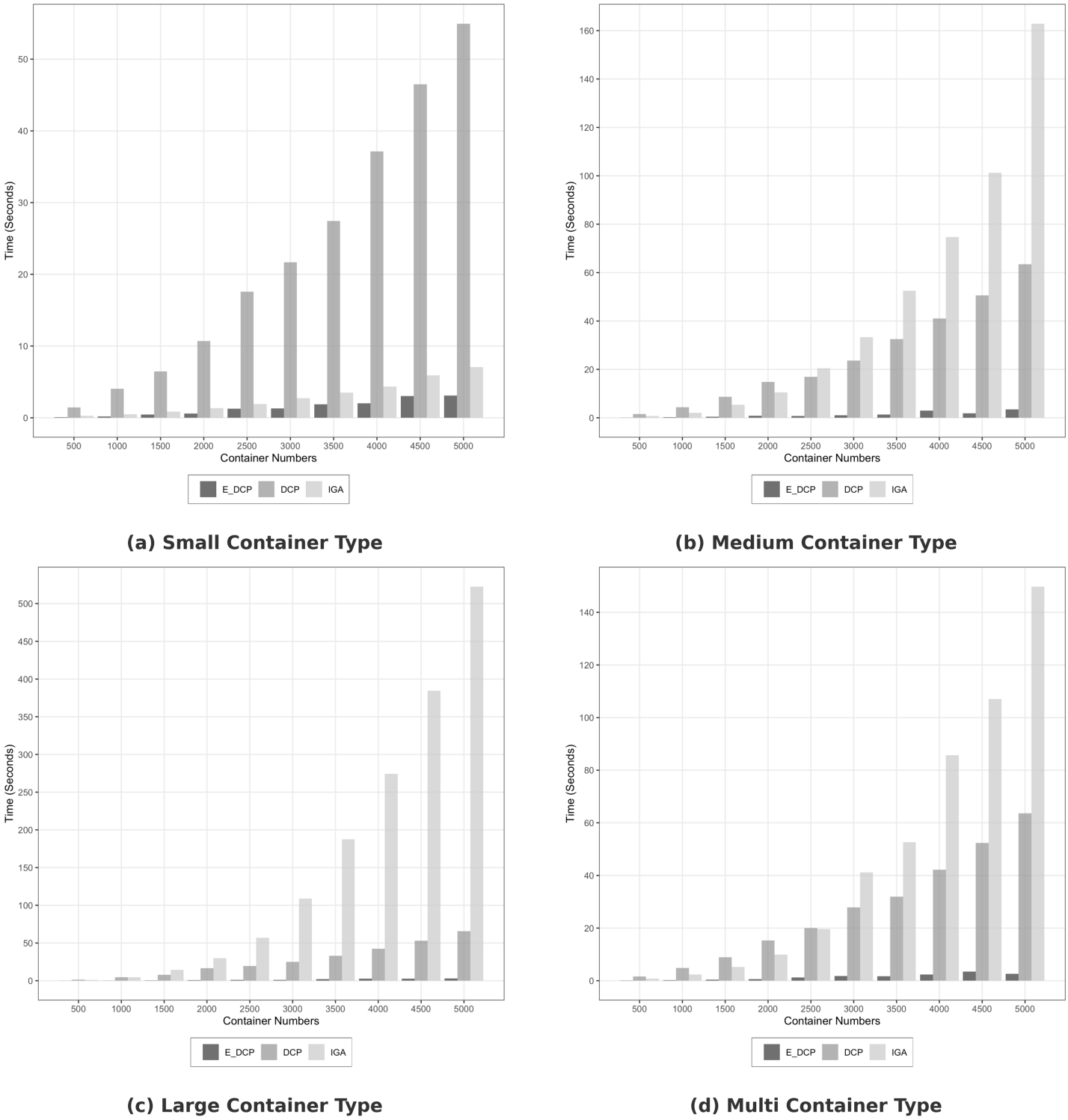

Figure 3 Search time in heterogeneous data center. (A) Small container type; (B) Medium container type; (C) Large container type; (D) Multi container type.
Full-size DOI: 10.7717/peerj-cs.3348/fig-3

Table 4 OLT results for small container type in homogeneous data centers.

Container #	E_DCP PM #	E_DCP OLT %	DCP PM #	DCP OLT %	IGA PM #	IGA OLT %
500	10	0%	10	0%	79	0%
1,000	19	0%	19	0%	65	0%
1,500	27	0%	27	0%	136	0%
2,000	37	0%	37	0%	200	0%
2,500	46	0%	46	0%	251	0%
3,000	55	0%	55	0%	186	0.53%
3,500	64	0%	64	0%	257	0%
4,000	73	0%	73	0%	266	0.37%
4,500	82	0%	83	0%	285	0.36%
5,000	93	0%	93	0%	308	0.3%

Table 5 OLT results for medium container type in homogeneous data centers.

Container #	E_DCP PM #	E_DCP OLT %	DCP PM #	DCP OLT %	IGA PM #	IGA OLT %
500	41	0%	44	9.09%	156	0%
1,000	78	0%	84	8.33%	279	0%
1,500	120	0%	129	5.43%	394	0.51%
2,000	165	0%	177	3.95%	482	2.07%
2,500	199	0%	213	9.86%	554	3.97%
3,000	246	0%	268	6.72%	634	5.52%
3,500	288	0%	310	4.84%	688	6.1%
4,000	331	0%	360	8.06%	746	8.18%
4,500	367	0%	397	6.55%	768	10.16%
5,000	411	0%	447	6.71%	819	16.24%

The first factor focuses on the level of service provided to cloud users while the second ensures the level of energy consumed is kept to the minimum. The utilization level in these experiments was set to 80% because several studies show that when CPU utilization consistently exceeds this level, the system may experience performance degradation (*Gregg, 2014*).

Tables 4, 5, 6 and 7 demonstrate the results of percentage of OLT PMs and number of active PMs of applying the mechanisms in homogeneous data centers. The data center consists of a PM with id 1 listed in Table 3. These tables demonstrate that the E_DCP mechanism outperformed both the DCP and IGA mechanisms in the number of OLT PMs by keeping the OLT level at 0%. The more containers to be processed the more PMs required to host them or the more specifications of these containers. For example, large container type requires more PMs than all other types provided that the number of containers remain the same. The E_DCP mechanism utilized less or similar number of PMs as in the DCP mechanism. For example, the number of used PMs was 416 for the E_DCP mechanism while it as 449 for the DCP mechanism when the container number

Table 6 OLT results for large container type in homogeneous data centers.

Container #	E_DCP PM #	E_DCP OLT %	DCP PM #	DCP OLT %	IGA PM #	IGA OLT %
500	75	0%	78	0%	249	0.4%
1,000	144	0%	153	2.61%	429	0.7%
1,500	222	0%	238	1.68%	576	3.13%
2,000	297	0%	317	0.63%	676	6.07%
2,500	359	0%	382	1.31%	752	10.51%
3,000	444	0%	473	1.69%	828	15.58%
3,500	511	0%	545	2.02%	870	23.01%
4,000	584	0%	622	2.73%	909	31.35%
4,500	657	0%	698	2.44%	935	40.75%
5,000	744	0%	795	2.39%	966	54.35%

Table 7 OLT results for multi container type in homogeneous data centers.

Container #	E_DCP PM #	E_DCP OLT %	DCP PM #	DCP OLT %	IGA PM #	IGA OLT %
500	43	0%	46	4.35%	168	0%
1,000	86	0%	93	3.23%	297	0%
1,500	124	0%	132	6.82%	395	0.25%
2,000	164	0%	179	3.91%	480	1.46%
2,500	205	0%	224	4.46%	565	2.65%
3,000	253	0%	274	5.47%	636	5.19%
3,500	294	0%	322	6.21%	692	7.23%
4,000	336	0%	365	5.48%	744	9.95%
4,500	367	0%	399	7.02%	778	11.57%
5,000	416	0%	449	7.8%	821	17.54%

was 5000 for multi container type in homogeneous data centers as it Table 7 depicts. The IGA mechanism yielded the worst results among others in both OLT number and number of active PMs.

Tables 8, 9, 10 and 11 reported the OLT results when employing the mechanisms for heterogeneous data centers. The E_DCP mechanism achieved minimum OLT level at 0% for all test runs except for container number was 5,000 for multi container type, the OLT level was just above 1%. However, the number of active PMs was higher than that of the DCP mechanism in most of the test runs. The increase of number of active PMs by the E_DCP mechanism in comparison to the DCP mechanism was between 0% to 13% while the increase of OLT PMs by the DCP mechanism in comparison to the E_DCP mechanism was between 0% to 50% in heterogeneous data centers. The experiments in both environments confirm that our proposed approach achieved much less number of OLT PMs but at a cost of more active PMs which may affect power consumption as it is discussed in the next section.

Table 8 OLT results for small container type in heterogeneous data centers.

Container #	E_DCP PM #	E_DCP OLT %	DCP PM #	DCP OLT %	IGA PM #	IGA OLT %
500	3	0%	2	50%	58	0%
1,000	4	0%	3	33.33%	62	0%
1,500	5	0%	5	0%	135	0%
2,000	6	0%	5	40%	138	0%
2,500	7	0%	7	57.14%	155	0%
3,000	9	0%	8	37.5%	211	0%
3,500	9	0%	9	44.44%	220	0%
4,000	11	0%	10	40%	240	0%
4,500	12	0%	11	45.45%	280	0%
5,000	13	0%	12	50%	311	0%

Table 9 OLT results for medium container type in heterogeneous data centers.

Container #	E_DCP PM #	E_DCP OLT %	DCP PM #	DCP OLT %	IGA PM #	IGA OLT %
500	8	0%	8	13%	170	0%
1,000	12	0%	12	17%	287	0%
1,500	19	0%	19	26%	399	0%
2,000	26	0%	23	39%	490	0%
2,500	37	0%	34	21%	565	0.71%
3,000	42	0%	40	40%	634	0.47%
3,500	50	0%	46	33%	684	0.88%
4,000	59	0%	53	34%	736	1.77%
4,500	64	0%	62	37%	765	1.96%
5,000	71	0%	65	37%	803	2.99%

Table 10 OLT results for large container type in heterogeneous data centers.

Container #	E_DCP PM #	E_DCP OLT %	DCP PM #	DCP OLT %	IGA PM #	IGA OLT %
500	13	0%	13	23.08%	256	0%
1,000	27	0%	26	26.92%	438	0%
1,500	45	0%	39	35.19%	570	0.53%
2,000	57	0%	53	30.19%	678	0%
2,500	70	0%	62	41.94%	757	1.59%
3,000	83	0%	78	37.18%	816	2.94%
3,500	95	0%	89	35.96%	861	2.56%
4,000	109	0%	101	36.63%	894	5.82%
4,500	121	0%	115	33.04%	927	5.29%
5,000	133	0%	126	34.92%	944	6.89%

Table 11 OLT results for multi container type in heterogeneous data centers.

Container #	E_DCP PM #	E_DCP OLT %	DCP PM #	DCP OLT %	IGA PM #	IGA OLT %
500	11	0%	11	18.18%	162	0%
1,000	19	0%	18	27.78%	284	0%
1,500	27	0%	26	23.08%	401	0%
2,000	36	0%	33	30.3%	479	0%
2,500	40	0%	39	23.08%	566	0.35%
3,000	50	0%	46	34.78%	646	0.62%
3,500	58	0%	52	40.38%	696	1.29%
4,000	66	0%	59	32.2%	737	2.17%
4,500	75	0%	67	35.82%	769	1.82%
5,000	84	1.19%	75	42.67%	812	1.72%

Energy efficiency

Energy efficiency is an important factor in cloud systems for two reasons. The first is to reduce running costs and the second is to reduced the negative impact of these systems on the the environment.

The results demonstrated that the proposed mechanism managed to consume less power in comparison to the DCP mechanism when it was applied to a homogeneous cloud system as it is depicted in Fig. 4. The worst scenario is when the container type is large and the number of containers is 5,000 because it will require more PMs to host high container requirements and large number of them. In this case, the E_DCP improve energy efficiency by about 5%. Similar improvements occurred in medium and multi container types. However, the energy consumed by our method is exactly the same as the DCP mechanism when the mechanisms were used in the small container type. This is because the requirements of this type of containers is quite low which leads to a similar behavior in these two mechanisms.

The total power consumed by a heterogeneous system yielded a different outcome. The DCP mechanism outperformed our mechanism by reducing consumption as it is illustrated in Fig. 5. This because our approach aims at reducing the number of OLT PMs which leads to more active PMs as it was explained in the previous subsection. The increase of active PMs means more energy consumption.

High service demand

Previous sections briefly outlined the implications of high service demand. This section provides a more rigorous analysis of the behavior of the E_DCP mechanism under such conditions. High service demand, defined as the increasing number of containers (*Alwabel, 2023*), poses a considerable challenge to system performance. As illustrated in Fig. 6A, the proposed mechanism achieves a substantial reduction in search time compared to the DCP and the IGA mechanisms, while sustaining comparable power consumption levels (Fig. 6B). Furthermore, Fig. 6C indicates that the E_DCP mechanism consistently

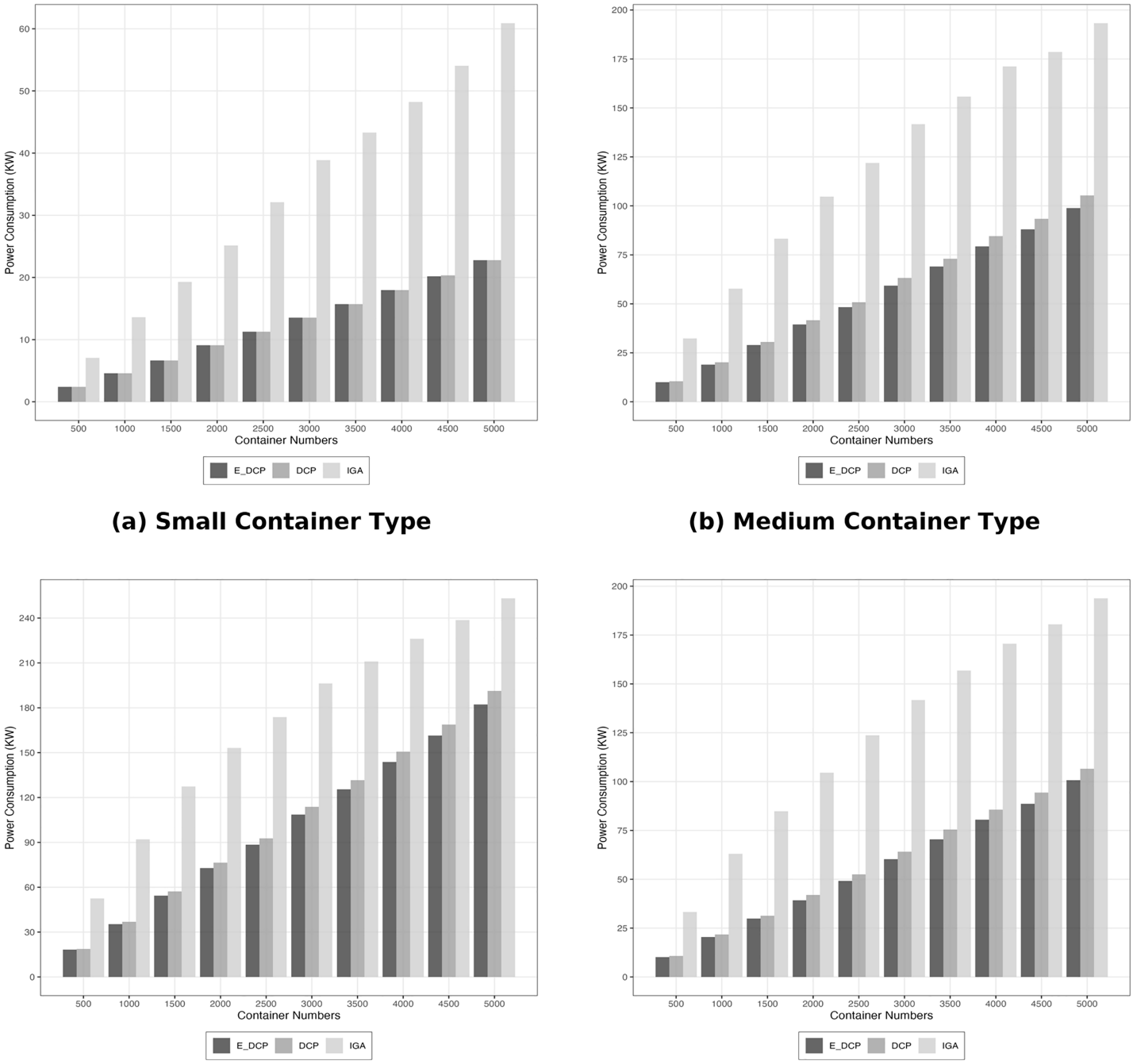

Figure 4 **Energy efficiency in homogeneous data center.** (A) Small container type; (B) Medium container type; (C) Large container type; (D) Multi container type.
Full-size DOI: 10.7717/peerj-cs.3348/fig-4

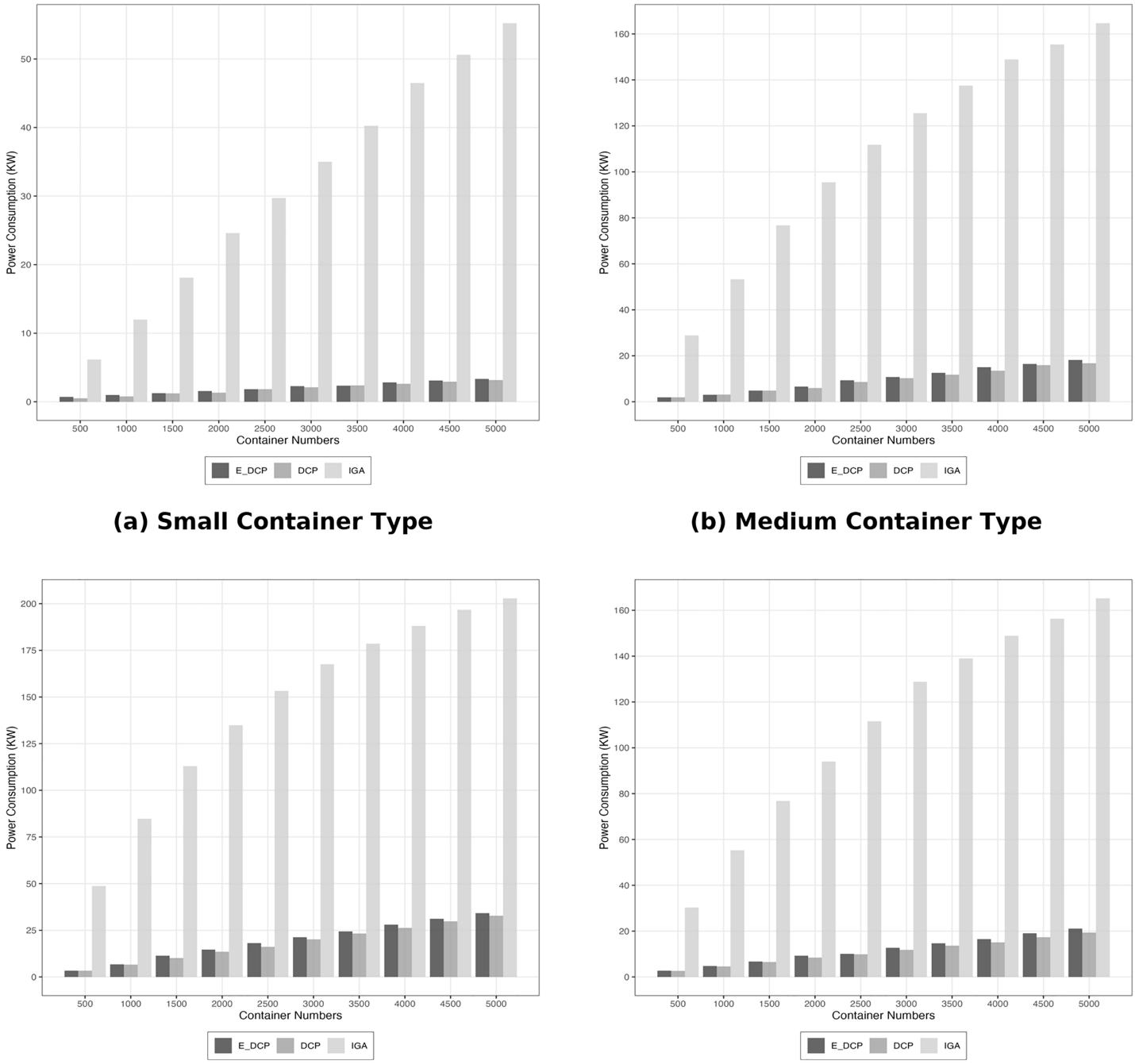

Figure 5 **Energy efficiency heterogeneous data center.** (A) Small container type; (B) Medium container type; (C) Large container type; (D) Multi container type.
Full-size DOI: 10.7717/peerj-cs.3348/fig-5

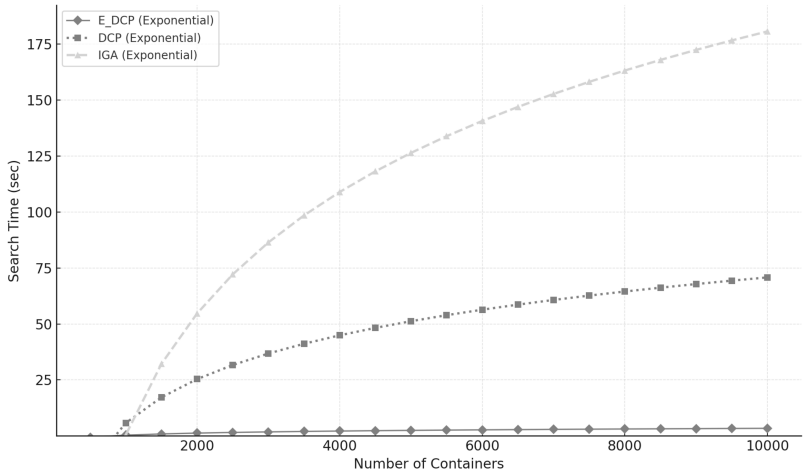

Figure 6 **Impact of high demand.** (A) Search time; (B) Power consumption; (C) OLT.
Full-size DOI: 10.7717/peerj-cs.3348/fig-6

maintains OLT levels at a minimum, highlighting its ability to ensure efficiency and scalability under intensive workloads.

CONCLUSION

Although containerized cloud services provide dynamic scalability and enhanced cost efficiency, it poses new challenges such as resource utilization and power consumption. This article presents a container placement mechanism that focuses on enhancing the performance of cloud systems by reducing search time and improving resource utilization while maintaining an acceptable level of power consumption in the case of large number of containers. The article employs WOA to find the best solution to place containers on PMs. The best solution is evaluated based on a scoring approach which this article discusses. Finally, the best solution is optimized to improve resource utilization with minimum affect on performance. The experiments demonstrate that our mechanism outperformed other mechanisms in terms of search time and resource utilization.

The E_DCP mechanism is evaluated in two cloud environments: homogeneous and heterogeneous. The experiments consider the large scale of workload (*i.e.*, the number of containers become quite high). The results demonstrated that the mechanism outperform the DCP and IGA mechanisms in both performance and resource utilization in both cloud environments. However, the price of this improvements is the increase of power consumption in comparisons to the DCP mechanism.

There are three major future research directions with this work. Firstly, the mechanism should consider resource availability as an evaluation criteria in the presence of failure events. In addition, a dynamic and migration approaches should be employed to further enhance the proposed method in terms of energy consumption. However, the overhead of migration should be taken into account in this study. Lastly, the evaluation approached presented in this article consider there factors and treat them equally. A work can investigate the effectiveness of assigning various weight for each factor in order to study if it can yield a better outcome.

ACKNOWLEDGEMENTS

The author acknowledges the use of ChatGPT, an AI language model, to review the language of certain sections of this article and to suggest relevant references. All final edits and decisions were undertaken by the author.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

This work was supported by the Prince Sattam bin Abdulaziz University through the project number (PSAU/2024/01/31506). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Grant Disclosures

The following grant information was disclosed by the authors:
Prince Sattam bin Abdulaziz University: PSAU/2024/01/31506.

Competing Interests

The authors declare that they have no competing interests.

Author Contributions

- Abdulelah Alwabel conceived and designed the experiments, performed the experiments, analyzed the data, performed the computation work, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.

Data Availability

The following information was supplied regarding data availability:

The code is available in the Supplemental File.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.3348#supplemental-information.

REFERENCES

Ahmad I, AlFailakawi MG, AlMutawa A, Alsalman L. 2022. Container scheduling techniques: a survey and assessment. *Journal of King Saud University—Computer and Information Sciences* **34(7)**:3934–3947 DOI 10.1016/j.jksuci.2021.03.002.

Ahmed J, Malik A, Ilyas MU, Alowibdi JS. 2019. Instance launch-time analysis of OpenStack virtualization technologies with control plane network errors. *Computing* **101(8)**:989–1014 DOI 10.1007/s00607-018-0626-5.

Al-Moalmi A, Luo J, Salah A, Li K, Yin L. 2021. A whale optimization system for energy-efficient container placement in data centers. *Expert Systems with Applications* **164(3)**:113719 DOI 10.1016/j.eswa.2020.113719.

Alahmad Y, Daradkeh T, Agarwal A. 2018. Availability-aware container scheduler for application services in cloud. In: *2018 IEEE 37th International Performance Computing and Communications Conference, IPCCC 2018*. Piscataway: IEEE.

Alwabel A. 2023. A novel container placement mechanism based on whale optimization algorithm for CaaS clouds. *Electronics* **12(15)**:3369 DOI 10.3390/electronics12153369.

Beloglazov A, Buyya R. 2012. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience* **24(13)**:1397–1420 DOI 10.1002/cpe.1867.

Bernstein D. 2014. Containers and cloud: from LXC to docker to kubernetes. *IEEE Cloud Computing* **1(3)**:81–84 DOI 10.1109/mcc.2014.51.

Buyya R, Yeo CS, Venugopal S. 2008. Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. In: *2008 10th IEEE International Conference on High Performance Computing and Communications*. Piscataway: IEEE, 5–13.

Çağlar I, Altılar DT. 2022. Look-ahead energy efficient VM allocation approach for data centers. *Journal of Cloud Computing* **11(1)**:11 DOI 10.1186/s13677-022-00281-x.

Carvalho M, Macedo DF. 2023. Container scheduling in co-located environments using QoE awareness. *IEEE Transactions on Network and Service Management* **20(3)**:3247–3260 DOI 10.1109/tnsm.2023.3244090.

Chuqiao Lin CL, Chuqiao Lin HS, Haoran Sun SW, Shengda Wang CA, Chunyan An HQ, Han Qi XL. 2023. Container migration strategy based on multi-objective optimization for edge-cloud coordination enabled smart grids. *Journal of Computers* **34(6)**:047–062 DOI 10.53106/199115992023123406004.

Deng S, Zhao H, Huang B, Zhang C, Chen F, Deng Y, Yin J, Dustdar S, Zomaya AY. 2024. Cloud-native computing: a survey from the perspective of services. *Proceedings of the IEEE* **112(1)**:12–46 DOI 10.36227/techrxiv.23500383.

Dorigo M, Birattari M, Stutzle T. 2006. Ant colony optimization. *IEEE Computational Intelligence Magazine* **1(4)**:28–39 DOI 10.1109/mci.2006.329691.

Gao Y, Guan H, Qi Z, Hou Y, Liu L. 2013. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences* **79(8)**:1230–1242 DOI 10.1016/j.jcss.2013.02.004.

Gill SS, Buyya R. 2019. A taxonomy and future directions for sustainable cloud computing. *ACM Computing Surveys* **51(5)**:1–33 DOI 10.1145/3241038.

Gregg B. 2014. *Systems performance: enterprise and the cloud*. First Edition. London, UK: Pearson Education, Inc.

Katal A, Dahiya S, Choudhury T. 2023. Energy efficiency in cloud computing data centers: a survey on software technologies. *Cluster Computing* **26(3)**:1845–1875 DOI 10.1007/s10586-022-03713-0.

Khan AA, Zakarya M, Buyya R, Khan R, Khan M, Rana O. 2019. An energy and performance aware consolidation technique for containerized datacenters. *IEEE Transactions on Cloud Computing* **9(4)**:1305–1322 DOI 10.1109/tcc.2019.2920914.

Lee EK, Viswanathan H, Pompili D. 2017. Proactive thermal-aware resource management in virtualized HPC cloud datacenters. *IEEE Transactions on Cloud Computing* **5(2)**:234–248 DOI 10.1109/tcc.2015.2474368.

Lin M, Xi J, Bai W, Wu J. 2019. Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. *IEEE Access* **7**:83088–83100 DOI 10.1109/access.2019.2924414.

Manvi SS, Krishna Shyam G. 2014. Resource management for infrastructure as a Service (IaaS) in cloud computing: a survey. *Journal of Network and Computer Applications* **41(1)**:424–440 DOI 10.1016/j.jnca.2013.10.004.

Mao Y, Oak J, Pompili A, Beer D, Han T, Hu P. 2017. DRAPS: dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster. In: *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. Piscataway: IEEE.

Merkel D. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* **2014(239)**.

Mirjalili S, Lewis A. 2016. The whale optimization algorithm. *Advances in Engineering Software* **95(12)**:51–67 DOI 10.1016/j.advengsoft.2016.01.008.

Mseddi A, Jaafar W, Elbiaze H, Ajib W. 2019. Joint container placement and task provisioning in dynamic fog computing. *IEEE Internet of Things Journal* **6(6)**:10028–10040 DOI 10.1109/jiot.2019.2935056.

Pahl C. 2015. Containerization and the PaaS cloud. *IEEE Cloud Computing* **2(3)**:24–31 DOI 10.1109/mcc.2015.51.

Pahl C, Brogi A, Soldani J, Jamshidi P. 2019. Cloud container technologies: a state-of-the-art review. *IEEE Transactions on Cloud Computing* **7(3)**:677–692 DOI 10.1109/tcc.2017.2702586.

Patra MK, Sahoo B, Turuk AK. 2024. FPHO: fractional pelican hawks optimization based container consolidation in CaaS cloud. *Concurrency and Computation: Practice and Experience* **36(12)**:e8052 DOI 10.1002/cpe.8052.

Piraghaj SF, Dastjerdi AV, Calheiros RN, Buyya R. 2015. 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*. Piscataway: IEEE, 368–375.

Randal A. 2020. The ideal versus the real: revisiting the history of virtual machines and containers. *ACM Computing Surveys* **53(1)**:1–31 DOI 10.1145/3365199.

Rosenblum M, Garfinkel T. 2005. Virtual machine monitors: current technology and future trends. *Computer* **38(5)**:39–47 DOI 10.1109/mc.2005.176.

Santos G, Paulino H, Vardasca T. 2020. QoE-aware auto-scaling 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, 242–249.

Shi T, Ma H, Chen G. 2018. Energy-aware container consolidation based on PSO in cloud data centers. In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. Piscataway: IEEE.

Wan X, Guan X, Wang T, Bai G, Choi B-Y. 2018. Application deployment using microservice and docker containers: framework and optimization. *Journal of Network and Computer Applications* **119(1)**:97–109 DOI 10.1016/j.jnca.2018.07.003.

Zhang R, Chen Y, Dong B, Tian F, Zheng Q. 2019. A genetic algorithm-based energy-efficient container placement strategy in CaaS. *IEEE Access* **7**:121360–121373 DOI 10.1109/access.2019.2937553.

Zhang D, Yan B-H, Feng Z, Zhang C, Wang Y-X. 2017. Container oriented job scheduling using linear programming model. In: *2017 3rd International Conference on Information Management (ICIM)*. Piscataway: IEEE, 174–180.

Zhao D, Mohamed M, Ludwig H. 2020. Locality-aware scheduling for containers in cloud computing. *IEEE Transactions on Cloud Computing* **8(2)**:635–646 DOI 10.1109/tcc.2018.2794344.

Zhong Z, Buyya R. 2020. A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources. *ACM Transactions on Internet Technology* **20(2)**:1–24 DOI 10.1145/3378447.

Zhou N, Zhou H, Hoppe D. 2023. Containerization for high performance computing systems: survey and prospects. *IEEE Transactions on Software Engineering* **49(4)**:2722–2740 DOI 10.1109/tse.2022.3229221.