electronics

MDPI

Article

A Novel Container Placement Mechanism Based on Whale Optimization Algorithm for CaaS Clouds

Abdulelah Alwabel

Department of Computer Sciences, Prince Sattam Bin Abdulaziz University, AlKharj 1194, Saudi Arabia; a.alwabel@psau.edu.sa

Abstract: Advancements in container technology can improve the efficiency of cloud systems by reducing the initiation time of virtual machines (VMs) and improving portability. Therefore, many cloud service providers offer cloud services based on the container as a service (CaaS) model. Container placement (CP) is a mechanism that allocates containers to a pool of VMs by mapping new containers to VMs and simultaneously considering VM placements on physical machines. The CP mechanism can serve several purposes, such as reducing power consumption and optimizing resource availability. This study presents directed container placement (DCP), a novel policy for placing containers in CaaS cloud systems. DCP extends the whale optimization algorithm, an optimization technique aimed at reducing the power consumption in cloud systems with a minimum effect on the overall performance. The proposed mechanism is evaluated against established methods, namely, improved genetic algorithm and discrete whale optimization using two criteria: energy savings and search time. The experiments demonstrate that DCP consumes approximately 78% less power and reduces the search time by approximately 50% in homogeneous clouds. In addition, DCP saves power by approximately 85% and reduces the search time by approximately 30% in heterogeneous clouds.

Keywords: cloud computing; CaaS; container placement; CP; energy efficiency

Citation: Alwabel, A. A Novel Container Placement Mechanism Based on Whale Optimization Algorithm for CaaS Clouds. *Electronics* **2023**, *12*, 3369. https://doi.org/10.3390/electronics12153369

Academic Editor: Mehdi Sookhak

Received: 5 July 2023
Revised: 31 July 2023
Accepted: 5 August 2023
Published: 7 August 2023

1. Introduction

Containerization is a relatively new technology for virtualizing applications in a lightweight manner and has led to significant utilization in cloud application management [1]. Cloud computing has become an efficient paradigm that offers computational abilities on a pay-per-usage basis [2]. Advancements in container technology can improve the efficiency of cloud systems by reducing the initiation time of virtual machines (VMs) and improving portability [3]. Consequently, popular cloud service providers, such as Amazon and Google, offer cloud services based on the container as a service (CaaS) model.

The term container refers to multi-tenant deployment techniques involving process isolation on a shared kernel to package an application and run it with isolated dependencies [4]. A container wraps a piece of software together with objects needed for the execution (i.e., runtime, libraries, and code) and permits easy deployment on any type of machine, whether physical machines (PMs) or VMs. Container technology exploits virtualization at the operating system (OS) level to take advantage of the flexibility and portability of software. A traditional VM requires all OS resources to be occupied [5], whereas a container can share the same OS kernel [6]. Therefore, a container consumes fewer resources and has a shorter deployment time because it is lightweight. However, it requires designing a feasible placement policy with optimal energy consumption under the adopted container-based cloud computing architecture [7].

Using containers with cloud computing promises to improve the portability between different cloud providers, which can help avert the risk of vendor locks [8]. Docker [9] and Kubernetes [10] are some of the most popular container solutions. They are built around

container engines in which a container acts as a portable means to package applications, resulting in the need to manage dependencies between containers in multi-tier applications. Figure 1 illustrates a cloud-based container architecture used to employ both VMs and container technology to provide specific requirements for each application. However, the VM layer can be eliminated while still providing a container for applications.

Container placement (CP) is a mechanism that allocates a list of containers to a pool of VMs by mapping new containers to VMs and simultaneously considering VM placements on the PMs [11]. The CP mechanism can help reduce power consumption and optimize resource availability. This paper presents directed container placement (DCP), a novel policy for CP in CaaS cloud systems. DCP employs an optimization technique with the aim of reducing power consumption in cloud systems with minimal effect on the overall performance. The mechanism is evaluated using two criteria: energy savings and search time. The remainder of this paper is organized as follows. Section 2 reviews the related work in the literature. Section 3 formulates the CP problem and Section 4 details the proposed mechanism. The experimental results are presented and discussed in Section 5. Section 6 summarizes the study and outlines future research directions.

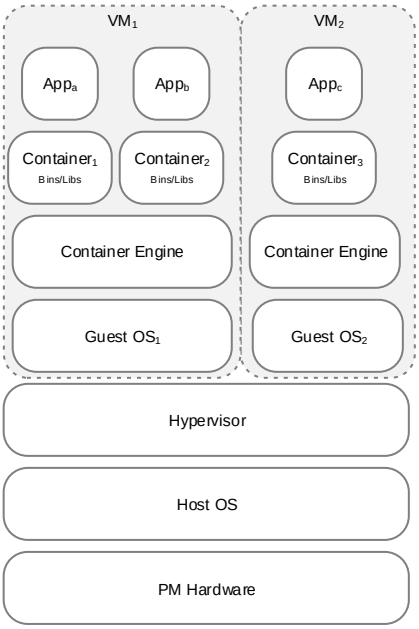

Figure 1. Container with VM architecture.

2. Literature Review

Cloud computing is a demanding technique. Over time, people are opting for digitization with the increasing use of smart applications and cloud services for everything. Therefore, clouds are moving to micro-clouding services to increase energy efficiency and minimize the burden on cloud data centers, owing to the increase in utilized resources. A container-based VM solution is a lightweight version that has recently been used to improve the performance of cloud services. This section highlights the contributions of previous studies with regards to the energy consumption, performance-aware, resource utilization and quality-aware mechanisms of container-based cloud systems as summarized in Table 1.

Table 1. Summary of reviewed papers.

Authors	Research Domain	Summary
Piraghaj et al. [12]	Power consumption	Framework for finding efficient size of VMs to host containers with minimum wastage of resources at VM
Kaur et al. [13]	Task selection and scheduling	CoESMS, a multi-objective function for reducing energy consumption by studying different limitations like CPU, memory, and worker budget
Shi et al. [14]	VM selection and placement optimization	Algorithm (called TMPSO) based on two phases to obtain energy-aware container consolidation in cloud data centers
Zhang et al. [15]	Energy and performance	Algorithm (called IGA) for efficiently searching the optimal CP solution
Al-Moalmi et al. [16]	Power consumption and resource utilization	Algorithm (called WOA) for power and resource utilization of containers and VM placements in CaaS environment
Zhang et al. [17]	Bi-objective optimization problem	FF-IGA algorithm for quick container positioning and enhancing the container managing cycle
Zhou et al. [18]	Container placement	Scheduling framework to leverage online primal-dual framework with a learning-based scheme for obtaining dual solutions
Boza et al. [19]	Lack of performance in cloud application	Performance-aware strategy for analyzing the performance of container applications with regards to runtime and initialization time
Luo et al. [20]	Power consumption	Task-planning algorithm for energy balancing of terminal devices to improve the life cycle of WSNs without expanding the postponement of tasks
Khan et al. [21]	Migration cost and consuming power	Combination of the migration (CPER) and the planned energy–performance-aware allocation (EPC-FU) methods to overcome migration cost, consume less power, examine energy-saving capacities, and present several workload types in data centers with containers
Nand et al. [22]	Resource optimization	Multi-resource bin filler algorithm (called RACC) that influences a deep learning method called fit-for-packing for attributing the near-optimal number of containers on PMs
Alahmad et al. [23]	Application service availability	Availability-aware container scheduling techniques to improve the accessibility of application services at end of cloud data center
Mseddi et al. [24]	Mobility/computation	Low-complexity, particle swarm optimization-based, meta-heuristic, and greedy heuristic algorithm developed for fog computing
Mendes et al. [25]	Resource and financial incentives	Docker swarm strategy for improving CPU and memory utilization over Spread and Binpack strategies
Zhong et al. [26]	Cost and performance	Resource utilization optimization and elastic instance pricing
Benomar et al. [27]	Virtualization	OpenStack-built middle-ware policy in which containers are positioned/achieved at the fog stages
Zhao et al. [28]	Load-balancing and application performance	Heuristic algorithms and methodology for different complex situations, for example, NP-hard
Nardelli et al. [29]	Container deployment problem	EVCD to regulate CP on computer-generated machines, which can be attained freely on demand while enhancing QoS measurements
Santos et al. [30]	QoE	Autonomic system (called ACTS) that vertically and horizontally measures a group of mixed containerized services exposed to various workloads with adaptation decisions depending on several high-level QoE metrics

2.1. Energy-Aware Mechanisms

The authors in [12] studied the CaaS environment model as a power optimization problem. Their proposed model also outperformed the energy efficiency issue in terms of CaaS via container association and reduced the number of active servers. They proposed an algorithm with correlation-aware placement, which showed that overload and underload threshold algorithms performed better than other algorithms when the selected container was larger with regards to migration. However, this work lacks studying and analyzing the proposed mechanism on different cloud environments, such as overloaded and underloaded data centers.

The authors in [13] proposed a container-based edge service management system (CoESMS), a multi-objective function for reducing energy consumption by studying different limitations, such as the CPU, memory, and worker budget. The results showed that the performance of the projected method was significantly improved, with a reduction in energy utilization of 21.75% and a decrease of 11.42% in service level agreement (SLA)

violations. General simulations were executed on the capacity traces obtained from Planet Lab. This work, however, can be further improved by considering resource utilization and throughput metrics in the mechanism.

An energy-alert model for merging optimized cloud containers was proposed in [14]. The algorithm was based on two phases and called multi-type particle swarm optimization (TMPSO). The results showed that the proposed algorithm could provide results in a more effective and efficient method, specifically for handling massive requests. The algorithm also showed additional energy savings compared with current approaches. This work can be extended by optimizing container consolidation during runtime.

The researchers in [15] presented the improved genetic algorithm (IGA), an enhanced genetic algorithm that works more efficiently in terms of energy savings for the CP problem. It operates more efficiently than the existing conventional GA, First-Fit, and particle swarm optimization (PSO). In conventional GA, a defect occurs, in which the variety of the population cannot be a guarantee as the utilization of resource VMs becomes complex. This problem was solved using IGA by optimizing the CP using a nonlinear energy consumption model. The results demonstrated that this technique can reduce the overall severe energy consumption. The proposed work, however, does not provide nor optimize the search time to find the best solution.

As it is becoming popular for data centers to use container services, deploying this service through cloud computing raises the hurdle of power consumption and resource utilization. In [16], the researchers proposed discrete whale optimization (DWO), which is a method to report the issue of the container and location of VM in CaaS situations, along with the idea of enhancing resource utilization and reducing power consumption based on the whale optimization algorithm (WOA) proposed by [31].

The authors planned a procedure built on WOA to solve the double phases of positioning as a single optimization issue. The results showed the superiority of the proposed technique over the evaluation approaches for a group of test situations. Therefore, the algorithm could be utilized to hold a greater number of containers or VMs because it decreased the number of PMs utilized. However, the proposed work can be extended to reduce the search time to find the best possible solution to reduce power consumption in CaaS cloud systems.

The authors in [17] developed a method for balancing the optimal performance service for the initial container placement in CaaS and the optimal power consumption. They designed a metric for the isolation application of UTS4 (a container-placing solution) based on anti-affinity limitations. This positioning optimization problem was then transformed to obtain the best solution for placement, along with low energy consumption and low UTS4. Additionally, a nonlinear power-consumption model was proposed. To find the placement solution, an algorithm was proposed and named the first-fit-based IGA algorithm (FF-IGA), which quickly places containers as an evolution of IGA to enhance the container management cycle. The results illustrated the effectiveness of the proposed algorithm and metric model by minimizing power consumption. However, mangy genetic-based approaches, including this work, endure a search time overhead, which negatively affects the overall performance of the systems.

2.2. Performance-Aware Mechanisms

The authors in [18] proposed a scheduling framework to leverage the online primal-dual framework with a learning-based scheme for obtaining dual solutions. It permits a job to state its job limit, inter-container, and selected cloud containers that influence the new scheduling algorithm strategy. They implemented a primal-dual model that shows the primal answer based on its double constraints in online and offline algorithms. The offline scheduling algorithm contains a new parting oracle to distinguish violated dual constraints. The online scheduling algorithm influences the online model based on a learning structure to obtain dual results. The results showed that the proposed scheduling frameworks were

more efficient and attained a close-to-ideal combined job estimation. However, the focus of this paper is on a different problem, which makes it out of the scope of this paper.

The authors in [19] targeted the problem of complexity in the operations of micro service storage. As cloud storage is increasing day by day, clouds are using these micro service containers to implement their applications. The aforementioned authors examined the initialization and runtime containerized performance applications together and reported that the default placement approach delivered by orchestrators is usually incomplete. Their performance-aware technique outperformed the default placement approach as shown in various experiments on multiple services. The performance placement strategy was varied up to $2x$ and $2.21x$ for the 50th and 99th percentiles, respectively.

The authors in [20] studied containers on a multi-cloud to multi-fog architecture and its related applications, attempting to address the task of personal cloud building. In addition, a task-planning algorithm built on energy balance was suggested to improve the lifetime of wireless sensor networks (WSNs) without expanding the postponement of tasks. They estimated the execution of virtual systems and containers under high-level concurrence, and the end effect proved that containers were improved compared with virtual devices. In addition, they created power expenses and task planning for the fog nodes and terminal devices (TDs). The proposed algorithm could efficiently stabilize the power of the TDs in a system while minimizing the service latency, which resulted in improving the performance of the system. Although this work managed to moderately consume power, it was not the main focus of the work to reduce power consumption.

In [21], to overcome the migration cost and consume less power, the authors suggested a combination of migration (CPER) and planned energy–performance-aware allocation (EPC-FU) methods to examine energy-saving capacities and the presentation of several workload types in data centers with containers. For a million containers, the overall top method bounded migrations to approximately 1.9% of the containers, of which the migrating cost recovered was 61.89%. The results showed that if more performance- and energy-effective hosts are migrated compared to containers that run for a long time, the economic feasibility of data centers is increased. However, the execution time of the proposed approach can negatively affect the QoS of CaaS cloud systems.

2.3. Resource Utilization Mechanisms

The authors in [22] proposed a learning approach called deep reinforcement to combine each of the functioning containers along with the various resource needs of at least the quantity of physical machinery. They presented and executed a multi-resource bin-filler algorithm (called RACC) that influences a deep learning method, called fit-for-packing, for attributing the near-optimal number of containers on physical machinery. The results showed that RACC attained an improved job slowdown compared with baseline algorithms. In addition, RACC expressed a considerable improvement in resource utilization. This work, however, does not consider power consumption as an evaluation metric of the proposed mechanism.

The authors in [23] proposed availability-aware container-scheduling techniques to improve the accessibility of application services at the end of a cloud data center. They planned to use physical computing and VMs for scheduling purposes with limitations because they have high availability values. The UML model was used for the computation. The CloudSim simulator was used to execute and integrate the strategies. The Docker container and the proposed strategy were compared in this study. Based on the results, other strategies had lower service availability than the proposed availability-aware strategy. In addition, this strategy obtained suitable host CPU utilization compared to other strategies.

In [24], to address the high computation cost, the authors combined container assignment and task provisioning in an active fog computing situation. Both the movement and irregular differences in the core function load were considered. The expansion problem of the number of appeals aided by the fog computing platform was expressed using integer linear programming. The results showed that the PSO-based algorithm achieved the best

results; however, the execution time was much longer than that of a greedy algorithm, which achieved up to 30% worse outcomes with no execution duration. In fact, this work focuses on the fog environment rather than cloud systems.

According to [25], the micro-cloud problem discussed in cloud computing is primarily moving towards micro storage. Thus, resources for devices are limited, with a lack of financial incentives for both the owner of the applications and the provider. By introducing a small overhead in arrangement times, their proposed solution accomplished the assignment of more requests, with an effective allocation of 83% in contradiction to 57% of prevalent answers, as measured on an arrangement of memory-concentrated workloads and a real CPU. However, this approach suffers from a drawback of causing an overhead of scheduling time.

The authors in [26] presented a heterogeneous task allocation strategy (HATS) for convenient and low-cost container orchestration via resource utilization optimization. HATS provided three features: (a) HATS initializes CP for optimal use by task packing; (b) using multiple auto-scaling algorithms, the authors proposed a pricing model of an elastic instance to adapt the cluster size with regards to the workload variation in runtime; and (c) they presented a rescheduling approach that uses the check pointing method of the container for permitting the cleaning of the VM of underused instances to save costs while maintaining task growth.

In [27], an OpenStack-built middleware policy was presented, in which containers can be positioned/achieved at the fog stages. The authors presented an S4T platform to provide an extension of the cloud for app designers and infrastructure executives. They presented a system proficient in handling distant containers that can collaborate and travel between various nodes without seeing the real outline of the structure. S4T offers a facility completely in agreement with OpenStack and cooperates with many of its facilities (e.g., Neutron and Keystone).

The conventional fog processing architecture is for a single data hub and several fog joints. It is incapable of adjusting to existing advancements in personal clouds. In [28], heuristic algorithms and answers were developed for different situations and many problems emerged as NP-hard problems. The planned method was applied and used for estimation on the de facto PaaS platform, namely, CloudFoundary. The results showed that the network load could be minimized by up to 60% with no harmful effect on its balance. On a large scale, they experimented with 2400 applications, achieving savings of approximately 90% in network traffic by removing 50% of the load balance. However, these two studied are not applicable for cloud systems because they were developed for fog systems.

2.4. Quality-Aware Mechanisms

The authors in [29] proposed a method called the elastic provisioning of VMs for container deployment (EVCD), which is an overall design for the placement of containers for cloud systems. EVCD regulates container placement on computer-generated machines that can be attained free on demand while enhancing the quality of service (QoS) measurements. In addition, it offers a standard next to which another container distribution heuristic can be matched. Accordingly, they estimated and emphasized the disadvantages of two well-known heuristics used to describe container placement.

The authors in [30] proposed an autonomic containerized service scaler (ACTS) system that vertically and horizontally measured a group of mixed containerized services exposed to various workloads to obtain results for several high-level QoE metrics. They employed ACTS in a few digitized facilities of the Shared Services of the Ministry of Health (SPMS) community corporation. The results showed that the proposed method could sufficiently adjust the configuration of all services as a direct response to modifications in its workload. The results also reduced cost and proved the capability of ACTS to retain the QoE system of measurement under restrictive standards pre-settled during SLAs. However, both EVCD and ACTS do not provide a means to reduce power consumption on CaaS cloud systems.

3. Problem Formulation

Server consolidation harnesses virtualization by sharing hardware to place multiple VMs on the same PM; therefore, fewer running PMs lead to greater power savings [32]. This section describes the objective model and corresponding constraints for the problem of placing containers efficiently to reduce power consumption. The power consumption of various PMs in data centers is calculated mainly based on the CPU, memory, disk storage, and networking, with the CPU consuming the most power [33]. CP refers to the containers hosted by VMs when VMs are placed on PMs in CaaS cloud systems [16]. This paper proposes a novel CP mechanism that improves the utilization of running PMs in cloud systems, leading to power savings.

Assuming that C is the set of containers to be assigned to V, V is the set of VMs to be run on P. Moreover, P is the set of available PMs in the CaaS data center. Let the resource specification of pm_i be defined as (pm_i^{cpu}, pm_i^{mem}), which represent the total processing and memory capacities, respectively, of pm_i. Let the tuple (vm_j^{cpu}, vm_j^{mem}) denote the processing and memory capacities, respectively, of vm_j. The resource requirements for the container c_k are denoted by (c_k^{cpu}, c_k^{mem}), which represent the processing and memory capacities, respectively, of c_k for $c_k \in C$. c_k can be assigned to vm_j as follows:

$$c_k^{cpu} \leq vm_j^{avl.cpu}, \forall\, c_k \in C\,,\ vm_j \in V \tag{1}$$

and

$$c_k^{mem} \leq vm_j^{avl.mem}, \forall\, c_k \in C\,,\ vm_j \in V \tag{2}$$

where $vm_j^{avl.cpu}$ and $vm_j^{avl.mem}$ denote the available processing and memory powers, respectively, of vm_j. This implies that containers can be placed on the same VM as long as the total processing or memory power of the containers does not exceed that of the VM. This can be formulated as $assign(c_k, vm_j)$, a function that returns true if vm_j hosts c_k or false otherwise, and is expressed as follows:

$$assign(c_k, vm_j) = \begin{cases} true,\ if\ equations \\ \quad (1)\ and\ (2)\ are\ true \\ false,\ otherwise \end{cases} \tag{3}$$

Moreover, vm_j is hosted on pm_i as follows:

$$vm_j^{cpu} \leq pm_i^{avl.cpu}, \forall\, vm_j \in V,\ pm_i \in P \tag{4}$$

and

$$vm_j^{mem} \leq pm_i^{avl.mem}, \forall\, vm_j \in V,\ pm_i \in P \tag{5}$$

where $pm_i^{avl.cpu}$ and $pm_i^{avl.mem}$ denote the available processing and memory, respectively, of pm_i. Let $allocate(vm_j, pm_i)$ be a function that returns true if vm_j can be allocated to pm_i or false otherwise. It can be expressed as follows:

$$allocate(vm_j, pm_i) = \begin{cases} true,\ if\ equations \\ \quad (4)\ and\ (5)\ are\ true \\ false,\ otherwise \end{cases} \tag{6}$$

This study adopts a linear server energy consumption model for containerized-based cloud systems [21]. The power consumption of pm_i is denoted by $pwr(pm_i)$ and is calculated as follows:

$$pwr(pm_i) = pm_i^{idle} + (pm_i^{max} - pm_i^{idle}) \times util(pm_i) \tag{7}$$

where pm_i^{max} is the power consumed by the PM when it runs at full utilization level and pm_i^{idle} is the power consumed by the PM when it is idle. An idle PM can consume an average of 70% of the power consumed by a machine at full utilization [34]. $utl(pm_i)$ denotes the current utilization level of pm_i, which is calculated as follows:

$$utl(pm_i) = \frac{pm_i^{cpu.used}}{pm_i^{cpu}} \tag{8}$$

where $pm_i^{cpu.used}$ denotes the total CPU power used for pm_i. In other words, this refers to the total processing power of all VMs hosted on pm_i. Let E_{cp} denote the total energy consumed by PMs using a particular CP in the CaaS cloud system, which is given as follows:

$$E_{cp} = \sum_{i=1}^{p} pwr(pm_i) \tag{9}$$

Therefore, a power-saving CP mechanism should aim at minimizing E_{cp}.

4. Proposed Mechanism

System Model

Figure 2 illustrates two mechanisms, namely, A and B, that can be adopted in CaaS cloud systems. Mechanism A does not focus on power consumption; therefore, it distributes the VMs around two different PMs, whereas Mechanism B places the three VMs on one PM if the PM can host all the VMs. Mechanism B can help reduce the overall power consumption provided that the second PM is in a power-saving mode. However, CP mechanisms can affect the overall performance of CaaS clouds because the overhead of finding an appropriate solution for CP can delay the processing of jobs in CaaS systems [35]. Therefore, the optimal mechanism should involve a trade-off between reducing the energy consumption and improving the system performance.

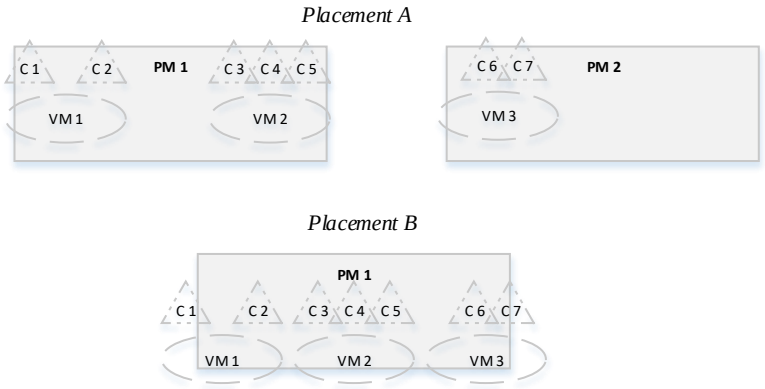

Figure 2. Different Placement Mechanisms.

This section presents the proposed DCP mechanism that extends the DWO mechanism [16], which is a container placement mechanism designed to improve resource utilization and reduce power consumption in CaaS-based cloud systems. DWO adopts the WOA [31], which is a meta-heuristic optimization algorithm that imitates the social behavior of humpback whales. A group of Humpback whales (i.e., search agents) make a spiral shape around the prey and then swim up to hunt the prey (i.e., a solution) inside the circles. Three phases are used by the humpback whale: (i) encircling, (ii) spiral bubble-net feeding, and (iii) searching for prey. The position of the prey is unknown; therefore, the current best search agent can be any position near the prey in the search space. Other search agents then update their positions according to the current search agent [16].

The algorithm employs a search process that commences with a randomly generated population of solutions (a matrix of solutions), which then evolves over successive generations. The solutions are always combined to form the next generation of solutions, allowing the matrix to be enhanced over the course of generations. However, the main difference between the current work and recently published papers in the literature (particularly the DWO mechanism [16]) is that the proposed DCP mechanism utilizes a heuristic approach to direct (hence, the name of the mechanism) the search process in the mechanism. This approach plays a vital role in reducing the overall power consumption of CaaS cloud systems with an acceptable overhead time.

The workflow of the DCP policy is depicted in Figure 3, which contains two phases: initialization and search. The initialization phase uses the number of PMs (p), VMs (v), and containers (c) as the input parameters. The mechanism also obtains the number of possible solutions (NW) and total number of iterations ($iterN$) to search for the best solution. The mechanism sets a as a double random number $\in [2, 0]$ that is linearly decreased from 2 to 0 [31]. The number of iterations t is set as zero. t is an iteration counter from one to $iterN$. Moreover, cf_1 and cf_2 are the coefficient vectors that are employed to determine how to update current solutions [31]. Those vectors are calculated as follows:

$$cf_1 = 2a \cdot rand - a \tag{10}$$

$$cf_2 = a \cdot rand \tag{11}$$

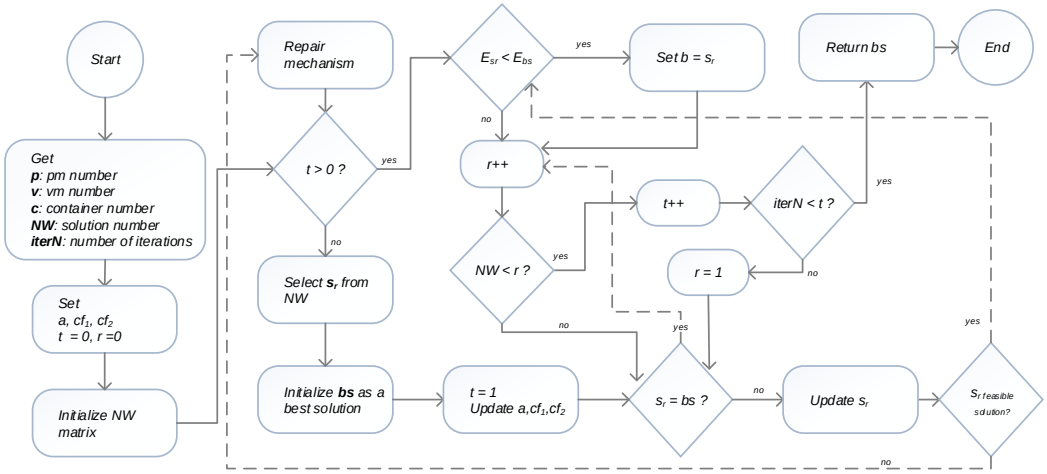

Figure 3. DCP flowchart.

Next, the policy initializes the solution matrix ($NWmatrix$) , which is a matrix of NW possible solutions, and each solution can be implemented using the CP mechanism in a cloud system. It randomly assigns containers to the VMs and allocates VMs to the PMs. Furthermore, the policy in the next step employs a repair mechanism that repairs the solutions in the $NWmatrix$ because the random allocation of containers to VMs can overload some VMs with containers, i.e., the CPU or RAM requirements of the containers are greater than those of a VM that hosts those containers. Similarly, some PMs can be overloaded with VMs. Overloaded VMs or PMs must be repaired by migrating containers and VMs from overloaded VMs and PMs, respectively.

The repair mechanism employs a two-factor approach to identify the most suitable containers (or VMs) for migration. Next, an appropriate candidate VM (or PM) is selected as the new host for these containers (or VMs) [16]. This approach aims to decrease the total power consumption by improving resource utilization. The first factor is the overloaded factor $OF(he, ge)$, which is a function employed to select a guest element ge. The guest

element can be a container or VM to be migrated from an overloaded hosting element *he*, which can be a VM or PM. The overloaded factor can be calculated as follows:

$$OF(he, ge) = \left| \frac{he_{cpu} - ge_{cpu}}{he_{cpu}} - \frac{he_{ram} - ge_{ram}}{he_{ram}} \right| \quad (12)$$

ge is a generic term that refers to either containers or VMs, and *he* is a generic term that refers to either VMs or PMs. The second factor is the selection factor $SF(he, ge)$, which is a function employed to select the VM (or PM) that is the most suitable candidate among all available choices. It can be calculated as follows:

$$SF(he, ge) = \left| \frac{he_{cpu} + ge_{cpu}}{he_{cpu}} - \frac{he_{ram} + ge_{ram}}{he_{ram}} \right| \quad (13)$$

Both Equations (12) and (13) are utilized to minimize the idle fragmentation of resources and help to efficiently harness the available resources to the maximum extent possible with an aim to improve resource utilization.

The repair mechanism fixes one solution at a time as shown in Algorithm 1. The mechanism input is a list of *he*. If *he* is overloaded, the mechanism retrieves all *ge* elements to select the elements to be migrated. The mechanism selects the *ge* element with the minimum *OF*. Subsequently, it removes it from the current *he* and adds it to the migration list as illustrated in lines 3–14 in the algorithm. Furthermore, the mechanism selects an appropriate destined host (*desHe*) for each *ge* in the migration list, *geMigrateList*. The *desHe* with the least *SF* is selected to host *ge* provided that *desHe* can host it as given in Equation (3) or Equation (6) if it is a VM or PM, respectively.

In the next step, DCP initializes the best solution (*bs*) by creating a solution that allocates VMs to the PMs based on the first-fit (FF) heuristic. The best solution initialization mechanism is presented in Algorithm 2. DCP employs the FF approach with the aim of reducing power consumption by stacking more VMs, which improves resource utilization. In contrast to other mechanisms based on the WOA, such as [36] and [16], DCP does not choose the best solution among the NW solutions in the solution matrix. Rather, it constructs a solution that can direct the search process, resulting in a reduced search time. The next step is to select the first solution s_r in the $NWmatrix$ and compare it with *bs* in terms of energy consumption, according to Equation (9). If s_r consumes less energy, it is selected as the new *bs*.

The search phase begins by updating the parameters a, cf_1, and cf_2 as mentioned in the initialization phase. Subsequently, if the current solution is not the same as the best solution, s_r is updated using the aforementioned parameters and *bs*, as explained in the WOA. Otherwise, DCP chooses the next solution in the matrix and continues the search process. The feasibility of the updated solution of s_r is checked to determine if it needs repair by sending it to the repair mechanism. Next, s_r is evaluated if it consumes less energy than *bs*; subsequently, *bs* becomes s_r as explained in the previous stage. The search phase then selects the next solution s_r in the solution matrix to repeat the same steps in this stage until the last solution in the matrix.

The entire search phase undergoes $iterN$ iterations, and each iteration is supposed to generate better solutions with *bs* being updated. Finally, DCP generates the optimal solution *bs*, which preserves the optimal energy in the CaaS cloud system under consideration.

Algorithm 1 Repair mechanism.

input: *heList*
foreach *ohe* in *heList* **do**
  while *ohe.isOverloaded* == *true* **do**
    $geList \leftarrow ohe.getGeList()$
    $OF = maximumValue$
    $migGe \leftarrow null$
    foreach *ge* in *geList* **do**
      $OF_{tmp} = OF(he, ge)$ //Equation (12)
      if $OF_{tmp} < OF$ **then**
        $OF = OF_{tmp}$
        $migGe \leftarrow ge$
      end if
      $ohe.removeElement(migGe)$
      $geMigrateList.add(migGe)$
    end for
  end while
end for
foreach *ge* in *geMigrateList* **do**
  $SF = maximumValue$
  $desHe \leftarrow null$
  foreach *he* in *heList* **do**
    if $he.canHost(ge) == true$ **then**
      $SFtmp = SF(he, ge)$ //Equation (13)
      if $SFtmp < SF$ **then**
        $SF = SFtmp$
        $desHe = he$
      end if
    end if
  end for
  $desHe.addElement(ge)$
  $geMigrateList.remove(ge)$
  $update(heList, desHe)$
end for

Algorithm 2 Best solution initialization.

input: *pmList*, *vmList*, *contList*
output: *bs*
foreach *continer* in *contList* **do**
  foreach *vm* in *pvmList* **do**
    if $assign(cont, vm) == true$ **then**
      $vm.assign(cont)$
    end if
  end for
end for
foreach *vm* in *vmList* **do**
  foreach *pm* in *pmList* **do**
    if $allocate(vm, pm) == true$ **then**
      $pm.allocate(vm)$
    end if
  end for
end for
$bs.setSolution(vmList, pmList)$
return *bs*

5. Performance Evaluation

5.1. Experimental Setup

The experiments in this study are performed on a computer with a 2.8 GHz Intel Core i7 processor and 16 GB of RAM with a MacOS Mojave OS. The experiments are conducted in the Java programming language. Two sets of experiments are conducted to evaluate the DCP policy. The first and second sets test the DCP policy in homogeneous and heterogeneous cloud systems, respectively.

Each experiment tests the mechanism for different container numbers: 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000. The DCP policy is based on the WOA, which produces solutions randomly; therefore, each experiment is run ten independent times for each number of containers, i.e., for number of containers = 100, the experiment is run ten times, then another ten times when the number of containers = 200 and so on. The total number of runs in each experimental set is 100. The reported results are the averages of the corresponding run results for each number of containers.

Two state-of-the-art container placement mechanisms are implemented for comparison to comprehensively evaluate the performance of DCP. The first is the IGA mechanism [15], which is a gene-based mechanism that utilizes different exchange mutation operations to determine the optimal solution. The second mechanism is the DWO mechanism [16], which is based on the WOA for solving the optimization problem of container-to-VM and VM-to-PM placement strategies. Both mechanisms are discussed in Section 2. They were selected because they both focus on energy consumption aspects in CaaS cloud systems. In addition, they both are evolution-based approaches, which make them suitable mechanisms to compare DCP with.

The search space for DCP, IGA, and WOA is set as 100. The number of solutions generated for IGA and WOA is set as 120 in each run. However, the number of solutions for the DCP policy is reduced to 30 because it decreases both the power consumption and execution time as demonstrated later in this section through the experiments.

The cloud system infrastructure in this simulation comprises PMs, VMs, and containers. The hardware specifications and power consumption values of the PMs are collected from the SPECpower benchmarks [37] as listed in Table 2. p_{idle} refers to the power consumed by a PM when it is idle, whereas p_{max} refers to the power consumed under full utilization. The number of available PMs is set as 1000. Various types of VMs are listed in Table 3, which represent several VM instances from Amazon [21]. Eight different types of containers are simulated in these experiments, each of which contains different CPU and RAM capabilities as summarized in Table 4. The cloud infrastructure values remain constant throughout the experiments.

Table 2. PM types.

PM Type	Model	No. of Cores	CPU (GHz)	RAM (GB)	p_{idle} (watt)	p_{max} (watt)
1	Fujitsu Primergy TX1310 M5	6	3.2	16	15.6	58.9
2	ProLiant DL20 Gen10 Plus	8	3.2	16	21.7	82.8
3	Uniwide RC2212	20	2.2	768	127	291
4	LTechKorea LKG-2212-C	32	2.9	512	96.6	377
5	Inspur NF5280M5	56	2.7	192	48.6	410
6	ProLiant DL325 Gen10 Plus	64	2.2	128	53.2	269

Table 3. VM types.

VM Type	Instance Model	No. of Cores	MIPS (GHz)	RAM (GB)
1	t2.nano	1	1.0	0.5
2	t1.micro	1	1.0	0.613
3	t2.micro	1	1.0	1.0
4	m1.small	1	1.0	1.7
5	m2.medium	2	2.0	3.75
6	m3.meduim	3	3.0	3.75

Table 4. Container types.

Container Type	MIPS (GHz)	RAM (GB)
1	0.256	0.128
2	0.512	0.128
3	1.0	0.128
4	0.256	0.256
5	0.512	0.256
6	1.0	0.256
7	0.512	0.512
8	1.0	0.512

5.2. Experiment I

The first experiment investigates the effectiveness of the DCP mechanism in a homogeneous cloud system. The simulated cloud system comprises PMs with the same hardware and power specifications. The hardware and power consumption values used in this experiment are those of the "Fujitsu Primergy TX1310 M5" model as listed in Table 2. Furthermore, one VM type is employed for this experiment, which is "m3.mediu" as listed in Table 3. Eight different types of containers are randomly employed as listed in Table 4.

The DCP mechanism outperforms both the IGA and DWO mechanisms in terms of power consumption. The average power consumption of the system is approximately 3100 W, 14,200 W, and 15,500 W for DCP, IGA, and DWO, respectively, as illustrated in Figure 4. Overall, the power consumed by the PMs for DCP decreases by approximately 78% compared with IGA; note that IGA yields better results on average than DWO. The effectiveness of the DCP mechanism improves as the number of containers increases as shown in Figure 5.

The DCP mechanism employs a resource utilization approach that aims to reduce the number of active PMs for placing VMs on PMs, leading to more power preservation. Therefore, more containers to run leads to a higher number of PMs. Table 5 lists the number of running VMs and PMs for each mechanism for each set of number of containers. For example, the DCP mechanism consumes approximately 800 W when the number of containers is 100, achieving approximately 53% and 70% power savings compared to IGA (1700 W) and DWO (2660) mechanisms, respectively. This is because the number of active PMs for DCP is six compared with 8 and 12 for the IGA and DWO mechanisms, respectively. The power savings increase sharply to more than ten-fold when the number of containers is 1000; to elaborate, the IGA and DWO mechanisms consume 30,000 W and 28,000 W, respectively, whereas DCP consumes approximately 5600 W.

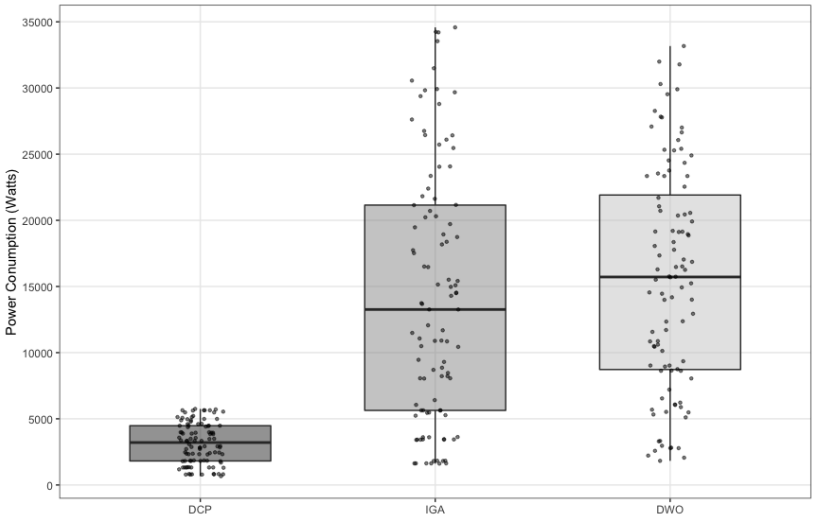

Figure 4. Power consumption in homogeneous environment.

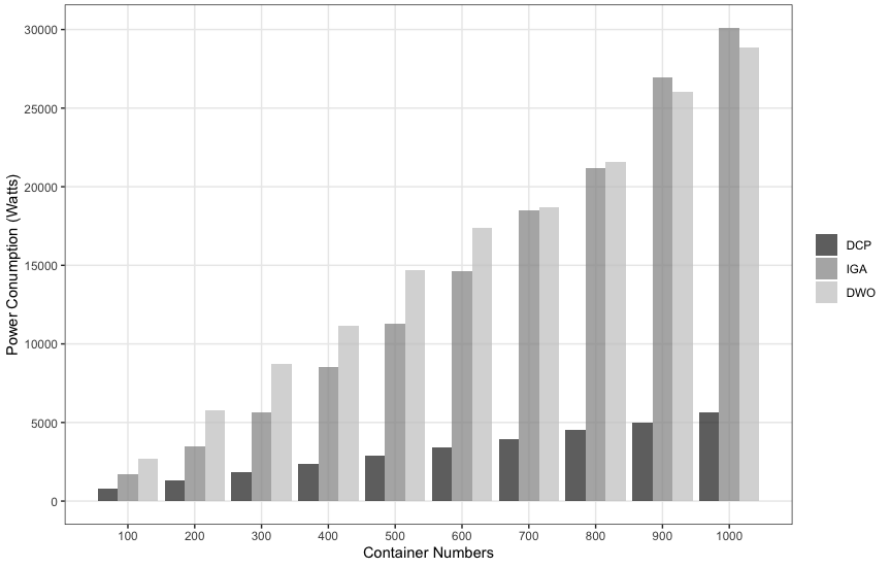

Figure 5. Power consumption per container number Set in homogeneous.

Table 5. Number of VMs and PMs in Homogeneous Environment.

	DCP		IGA		DWO	
Cont #	**VM #**	**PM #**	**VM #**	**PM #**	**VM #**	**PM #**
100	22	6	10	8	23	12
200	42	11	19	17	44	28
300	62	16	32	28	65	42
400	84	21	47	42	87	53
500	104	26	62	56	108	70
600	126	32	81	72	130	83
700	146	37	103	92	151	88
800	167	42	119	105	172	102
900	187	47	155	133	194	124
1000	210	53	172	149	218	137

The search times for each mechanism are shown in Figure 6. The DCP mechanism outperforms its counterparts in terms of performance. The results show that the DWO mechanism behaves poorly in terms of searching for the optimal solution as the number of containers increases. Although the IGA mechanism yields far better outcomes in terms of search time than DWO, it does not meet the performance of the DCP mechanism because DCP requires approximately 17 s on average to find the optimal solution, whereas this figure doubles to approximately 36 s for IGA. The search time for DWO is approximately 213 s on average. Overall, DCP demonstrably improves the performance of the container placement mechanisms by approximately 50%.

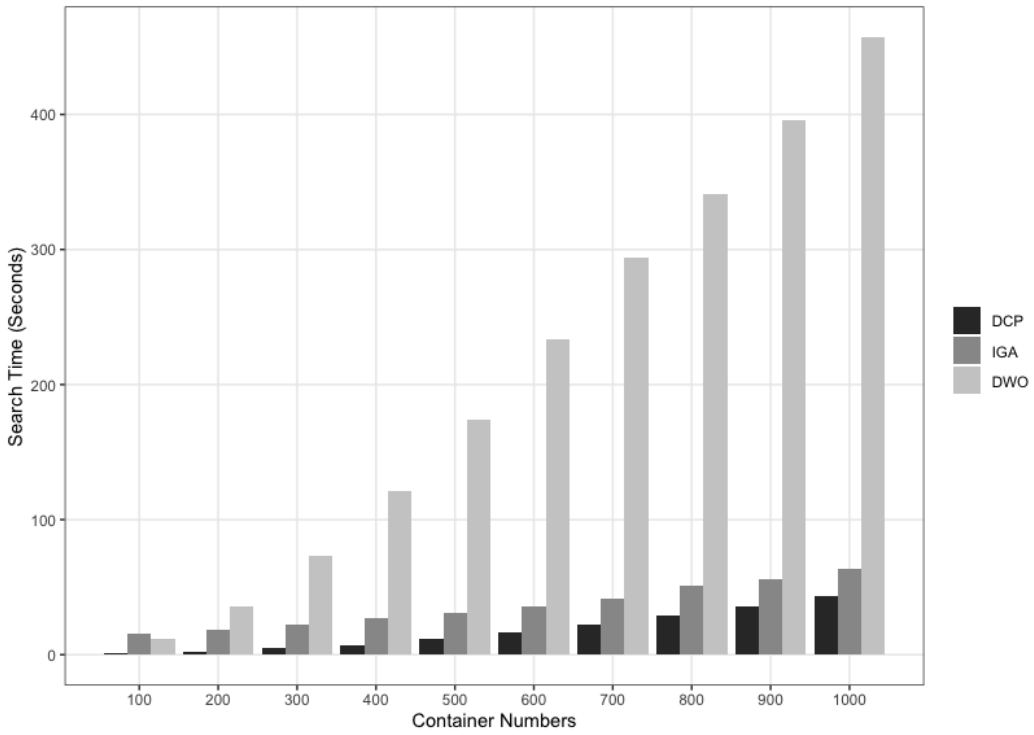

Figure 6. Search time in homogeneous environment.

5.3. Experiment II

The second experiment investigates the behavior of the DCP mechanism in heterogeneous cloud systems. In such an environment, PMs have various hardware and power consumption specifications, which is a more realistic assumption than that of homogeneous cloud systems. The power consumption results are depicted in Figure 7. The average power consumption for the DCP mechanism is approximately 1500 W, whereas the power consumption for the IGA and DWO mechanisms is approximately 33,000 W and 10,000 W, respectively. DCP saves power by approximately 85% compared to the other mechanisms. The effectiveness of the DCP mechanism improves as the number of containers increases as shown in Figure 8.

Table 6 lists the numbers of active VMs and PMs for each mechanism according to each container number set. Note that both DCP and DWO achieve better power consumption figures comparable to those for the same mechanisms from the first experiment because these mechanisms can find better solutions to reduce power consumption when different PM and VM specifications are employed. The greater the number of hosted containers, the greater the effectiveness of DCP. When the number of containers is 1000, the power consumed by DCP is 2700 W, whereas DWO consumes approximately 14,300 W.

The search time for DCP is approximately 21 s on average, whereas the average time required to find the optimal solution for IGA is approximately 69 s and that for IGA is approximately 212 s. DWO preserves power better than IGA but at the expense of

performance. DCP manages to reduce power consumption and improve performance in comparison with other mechanisms. Figure 9 presents the average search time for each mechanism. More containers lead to more searching, resulting in more time being required to find the best solution. For example, DCP requires, on average, approximately 1 s to find the best solution when the number of containers is 100; however, the time required increases to just more than 50 s when the number of containers is 1000.

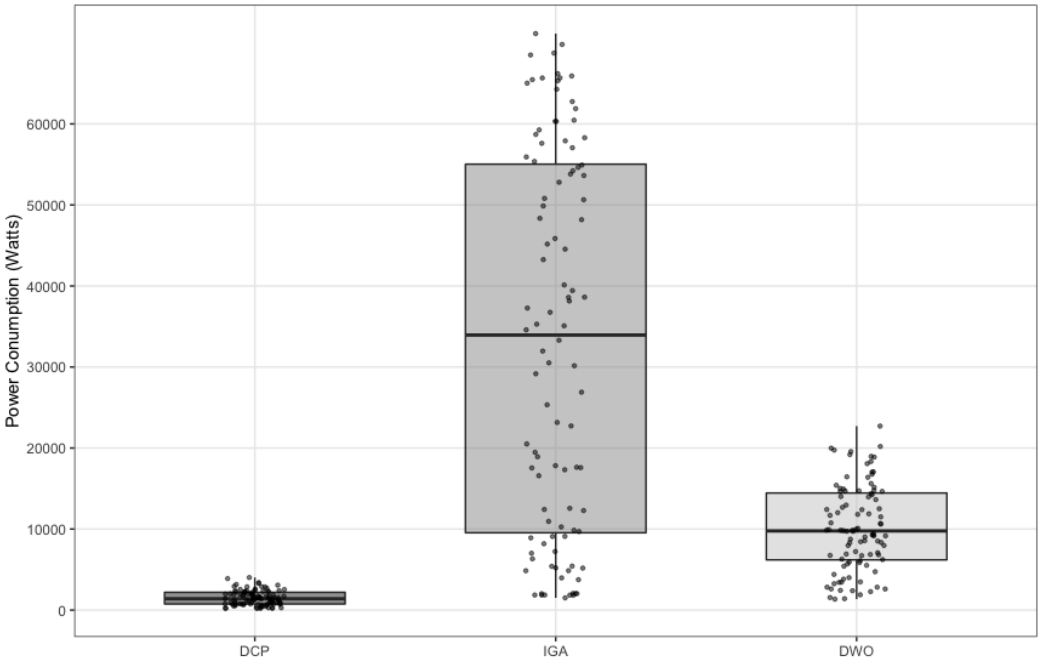

Figure 7. Power consumption in heterogeneous environment.

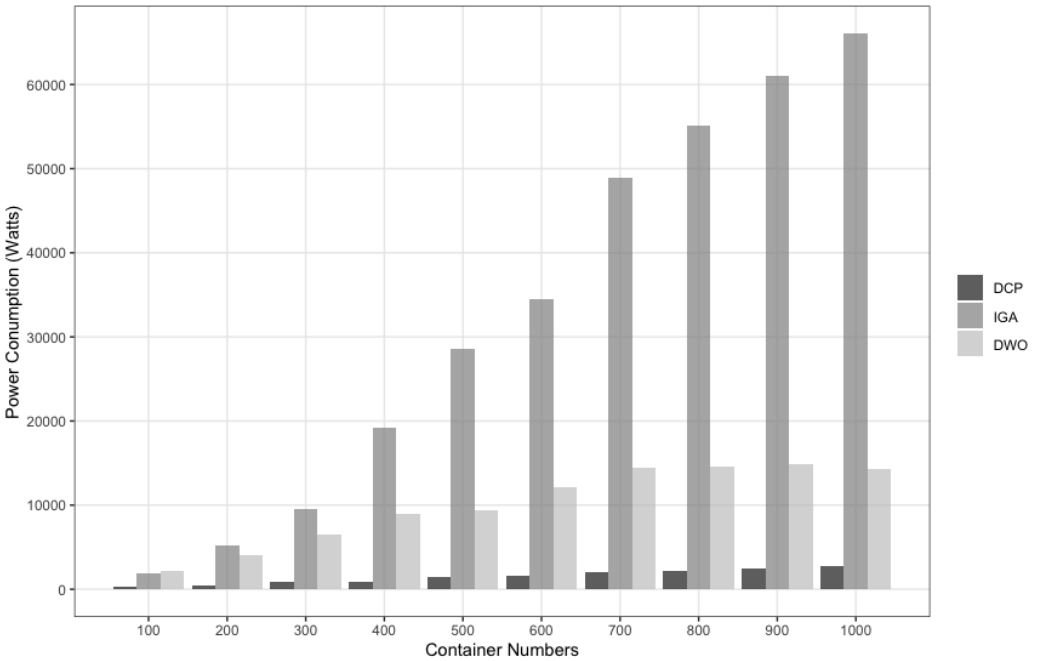

Figure 8. Power consumption per container number set in heterogeneous environment.

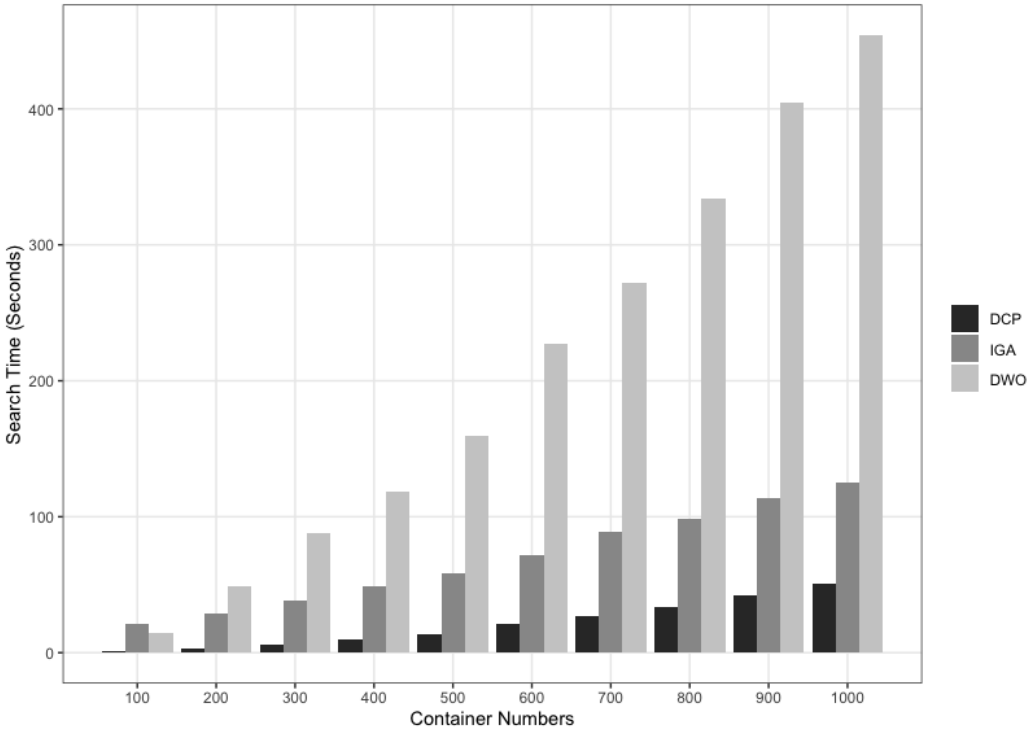

Figure 9. Search time in heterogeneous environment.

Table 6. Number of VMs and PMs in heterogeneous environment.

	DCP		IGA		DWO	
Cont #	**VM #**	**PM #**	**VM #**	**PM #**	**VM #**	**PM #**
100	41	2	11	10	39	12
200	87	2	30	28	81	22
300	127	3	56	52	120	34
400	170	3	116	104	163	47
500	213	6	179	154	203	49
600	256	6	220	186	237	63
700	306	8	333	264	292	76
800	340	9	385	297	332	76
900	386	10	433	329	372	76
1000	428	10	484	356	414	73

This section demonstrates that the proposed mechanism can save energy in both homogeneous and heterogeneous cloud systems. DCP can perform better in heterogeneous systems because the mechanism employs a resource utilization approach that focuses on reducing the number of active PMs. However, the mechanism does not focus on reducing the number of active VMs compared with other related mechanisms. Furthermore, the performance of DCP is better than that of the alternatives because it reduces the time required to find the optimal solution in both homogeneous and heterogeneous environments.

6. Conclusions

CP mechanisms are critical in CaaS cloud systems with regards to energy savings. This study presents the DCP mechanism, a novel CP policy that significantly reduces power consumption. DCP extends the WOA technique to find the best solution in a relatively short time. DCP is evaluated and compared with the IGA and DWO mechanisms in two different cloud systems: homogeneous and heterogeneous. The experiments demonstrate that DCP

consumes approximately 78% less power while reducing the search time by approximately 50% in homogeneous clouds. In addition, DCP saves power by approximately 85% while reducing the search time by approximately 30% in heterogeneous clouds.

Considering the scope for future research, the DCP mechanism can be improved as follows. First, it can consider more optimization objectives, such as high availability and low resource wastage. Furthermore, DCP can be extended to reduce the number of VMs assigned to containers. Second, the evaluation conducted in this study uses simulations with randomly generated containers, VMs, and PMs. Therefore, the proposed mechanism can be further tested on real systems. Third, the DCP policy is a static placement mechanism. Thus, it can be extended to be a dynamic mechanism with the aim of rescheduling containers during runtime. Finally, the DCP mechanism can be extended and evaluated for cloud systems integrated with fog systems.

Funding: This project was supported by the Deanship of Scientific Research at Prince Sattam Bin Abdulaziz University under research project No. 2018/01/9371.

Data Availability Statement: No data were used to support this study.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Pahl, C.; Brogi, A.; Soldani, J.; Jamshidi, P. Cloud container technologies: A state-of-the-art review. *IEEE Trans. Cloud Comput.* **2019**, *7*, 677–692. [CrossRef]
2. Buyya, R.; Ranjan, R.; Calheiros, R.N. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In Proceedings of the 2009 International Conference on High Performance Computing & Simulation, Leipzig, Germany, 21–24 June 2009; pp. 1–11. [CrossRef]
3. Morabito, R. Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation. *IEEE Access* **2017**, *5*, 8835–8850. [CrossRef]
4. Randal, A. The Ideal Versus the Real: Revisiting the History of Virtual Machines and Containers. *ACM Comput. Surv.* **2020**, *53*, 1–31. [CrossRef]
5. Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield, A. Xen and the art of virtualization. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles-SOSP '03, New York, NY, USA, 19–22 October 2003; p. 164. [CrossRef]
6. Bernstein, D. Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Comput.* **2014**, *1*, 81–84. [CrossRef]
7. Piraghaj, S.F.; Dastjerdi, A.V.; Calheiros, R.N.; Buyya, R. Efficient Virtual Machine Sizing for Hosting Containers as a Service (SERVICES 2015). In Proceedings of the 2015 IEEE World Congress on Services, New York, NY, USA, 27 June–2 July 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 31–38. [CrossRef]
8. Linthicum, D.S. Moving to Autonomous and Self-Migrating Containers for Cloud Applications. *IEEE Cloud Comput.* **2016**, *3*, 6–9. [CrossRef]
9. Merkel, D. Docker Lightweight Linux Containers for Consistent Development and Deployment *J. Linux.* **2014**, *239*, 2.
10. Hightower, K.; Burns, B.; Beda, J. *Kubernetes: Up and Running: Dive into the Future of Infrastructure*, 1st ed.; O'Reilly Media Inc.: Sebastopol, CA, USA , 2017; p. 272.
11. Hussein, M.K.; Mousa, M.H.; Alqarni, M.A. A placement architecture for a container as a service (CaaS) in a cloud environment. *J. Cloud Comput.* **2019**, *8*, 7. [CrossRef]
12. Piraghaj, S.F.; Dastjerdi, A.V.; Calheiros, R.N.; Buyya, R. A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers. In Proceedings of the 2015 IEEE International Conference on Data Science and Data Intensive Systems, Sydney, NSW, Australia, 11–13 December 2015; IEEE; Piscataway, NJ, USA, 2015; pp. 368–375. [CrossRef]
13. Kaur, K.; Dhand, T.; Kumar, N.; Zeadally, S. Container-as-a-Service at the Edge: Trade-off between Energy Efficiency and Service Availability at Fog Nano Data Centers. *IEEE Wirel. Commun.* **2017**, *24*, 48–56. [CrossRef]
14. Shi, T.; Ma, H.; Chen, G. Energy-Aware Container Consolidation Based on PSO in Cloud Data Centers. In Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 8–13 July 2018; IEEE; Piscataway, NJ, USA, 2018; pp. 1–8. [CrossRef]
15. Zhang, R.; Chen, Y.; Dong, B.; Tian, F.; Zheng, Q. A Genetic Algorithm-Based Energy-Efficient Container Placement Strategy in CaaS. *IEEE Access* **2019**, *7*, 121360–121373. [CrossRef]
16. Al-Moalmi, A.; Luo, J.; Salah, A.; Li, K.; Yin, L. A whale optimization system for energy-efficient container placement in data centers. *Expert Syst. Appl.* **2021**, *164*, 113719. [CrossRef]
17. Zhang, R.; Chen, Y.; Zhang, F.; Tian, F.; Dong, B. Be Good Neighbors: A Novel Application Isolation Metric Used to Optimize the Initial Container Placement in CaaS. *IEEE Access* **2020**, *8*, 178195–178207. [CrossRef]

18. Zhou, R.; Li, Z.; Wu, C. Scheduling Frameworks for Cloud Container Services. *IEEE/ACM Trans. Netw.* **2018**, *26*, 436–450. [CrossRef]
19. Boza, E.F.; Abad, C.L.; Narayanan, S.P.; Balasubramanian, B.; Jang, M. A case for performance-aware deployment of containers. In Proceedings of the WOC 2019—Proceedings of the 2019 5th International Workshop on Container Technologies and Container Clouds, Part of Middleware 2019, Davis, CA, USA, 9–13 December 2019; pp. 25–30. [CrossRef]
20. Luo, J.; Yin, L.; Hu, J.; Wang, C.; Liu, X.; Fan, X.; Luo, H. Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT. *Future Gener. Comput. Syst.* **2019**, *97*, 50–60. [CrossRef]
21. Khan, A.A.; Zakarya, M.; Buyya, R.; Khan, R.; Khan, M.; Rana, O. An energy and performance aware consolidation technique for containerized datacenters. *IEEE Trans. Cloud Comput.* **2019**, *9*, 1305–1322. [CrossRef]
22. Nanda, S.; Hacker, T.J. RACC: Resource-Aware Container Consolidation using a Deep Learning Approach. In Proceedings of the First Workshop on Machine Learning for Computing Systems, New York, NY, USA, 12 June 2018; pp. 1–5. [CrossRef]
23. Alahmad, Y.; Daradkeh, T.; Agarwal, A. Availability-Aware Container Scheduler for Application Services in Cloud. In Proceedings of the 2018 IEEE 37th International Performance Computing and Communications Conference, IPCCC 2018, Orlando, FL, USA, 17–19 November 2018. [CrossRef]
24. Mseddi, A.; Jaafar, W.; Elbiaze, H.; Ajib, W. Joint Container Placement and Task Provisioning in Dynamic Fog Computing. *IEEE Internet Things J.* **2019**, *6*, 10028–10040. [CrossRef]
25. Mendes, S.; Simão, J.; Veiga, L. Oversubscribing micro-clouds with energy-aware containers scheduling. In Proceedings of the ACM Symposium on Applied Computing, Limassol, Cyprus, 8–12 April 2019; Part F1477, pp. 130–137. [CrossRef]
26. Zhong, Z.; Buyya, R. A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources. *ACM Trans. Internet Technol.* **2020**, *20*, 1–24. [CrossRef]
27. Benomar, Z.; Longo, F.; Merlino, G.; Puliafito, A. Cloud-based Enabling Mechanisms for Container Deployment and Migration at the Network Edge. *ACM Trans. Internet Technol.* **2020**, *20*, 1–28. [CrossRef]
28. Zhao, D.; Mohamed, M.; Ludwig, H. Locality-Aware Scheduling for Containers in Cloud Computing. *IEEE Trans. Cloud Comput.* **2020**, *8*, 635–646. [CrossRef]
29. Nardelli, M.; Hochreiner, C.; Schulte, S. Elastic Provisioning of Virtual Machines for Container Deployment. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, New York, NY, USA, 22–26 April 2017; pp. 5–10. [CrossRef]
30. Santos, G.; Paulino, H.; Vardasca, T. 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, 30 March–3 April 2020; pp. 242–249. [CrossRef]
31. Mirjalili, S.; Lewis, A. The Whale Optimization Algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [CrossRef]
32. Berl, a.; Gelenbe, E.; Di Girolamo, M.; Giuliani, G.; De Meer, H.; Dang, M.Q.; Pentikousis, K. Energy-Efficient Cloud Computing. *Comput. J.* **2010**, *53*, 1045–1051. [CrossRef]
33. Beloglazov, A.; Buyya, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurr. Comput. Pract. Exp.* **2012**, *24*, 1397–1420. [CrossRef]
34. Kusic, D.; Kephart, J.O.; Hanson, J.E.; Kandasamy, N.; Jiang, G. Power and performance management of virtualized computing environments via lookahead control. *Clust. Comput.* **2008**, *12*, 1–15. [CrossRef]
35. Ahmad, I.; AlFailakawi, M.G.; AlMutawa, A.; Alsalman, L. Container scheduling techniques: A Survey and assessment. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 3934–3947. [CrossRef]
36. Vhatkar, K.N.; Bhole, G.P. Optimal container resource allocation in cloud architecture: A new hybrid model. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 1906–1918. [CrossRef]
37. Lange, K.D. Identifying Shades of Green: The SPECpower Benchmarks. *Computer* **2009**, *42*, 95–97. [CrossRef]