Hussein *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2019) 8:7
https://doi.org/10.1186/s13677-019-0131-1

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

A placement architecture for a container as a service (CaaS) in a cloud environment

Mohamed K. Hussein[1*], Mohamed H. Mousa[1,2] and Mohamed A. Alqarni[2]

Abstract

Unlike a traditional virtual machine (VM), a container is an emerging lightweight virtualization technology that operates at the operating system level to encapsulate a task and its library dependencies for execution. The Container as a Service (CaaS) strategy is gaining in popularity and is likely to become a prominent type of cloud service model. Placing container instances on virtual machine instances is a classical scheduling problem. Previous research has focused separately on either virtual machine placement on physical machines (PMs) or container, or only tasks without containerization, placement on virtual machines. However, this approach leads to underutilized or overutilized PMs as well as underutilized or overutilized VMs. Thus, there is a growing research interest in developing a container placement algorithm that considers the utilization of both instantiated VMs and used PMs simultaneously.

The goal of this study is to improve resource utilization, in terms of number of CPU cores and memory size for both VMs and PMs, and to minimize the number of instantiated VMs and active PMs in a cloud environment. The proposed placement architecture employs scheduling heuristics, namely, Best Fit (BF) and Max Fit (MF), based on a fitness function that simultaneously evaluates the remaining resource waste of both PMs and VMs. In addition, another meta-heuristic placement algorithm is proposed that uses Ant Colony Optimization based on Best Fit (ACO-BF) with the proposed fitness function. Experimental results show that the proposed ACO-BF placement algorithm outperforms the BF and MF heuristics and maintains significant improvement of the resource utilization of both VMs and PMs.

Keywords: Cloud container, Virtual machine placement, Container placement, Two-tier placement algorithm, Ant colony, Best fit, Max fit

Introduction

Cloud computing is the on-demand delivery of computing applications, storage and infrastructure as services provided to customers over the Internet. Users pay only for the services that they use according to a utility charging model [1–3]. Cloud services are typically provided in three different service models: (1) Infrastructure as a Service (IaaS), which is a computing infrastructure in which storage and computing hardware are provided through API and web portals; (2) Platform as a Service (PaaS), which is a platform for application development provided as services, such as scripting environments and database services. (3) Software as a Service (SaaS), which provides applications as services, (e.g. Google services).

Virtual machine (VM) technology is the leading execution environment used in cloud computing. The resources of a physical machine (PM), such as CPUs, memory, I/O, and bandwidth, are partitioned among independent virtual machines, each of which runs an independent operating system. A Virtual Machine Monitor (VMM), also called a hypervisor, manages and controls all instantiated VMs, as shown in Fig. 1a.

A container is a new lightweight virtualization technology that does not require a VMM. A container provides an isolated virtual environment at the operating system level. Furthermore, different containers executing on a specific operating system share the same operating system kernel; consequently, executing a container does not require launching an operating system, as shown in Fig. 1b.

* Correspondence: m_khamiss@ci.suez.edu.eg
[1]Faculty of Computers and Informatics, Suez Canal University, Ismailia, Egypt
Full list of author information is available at the end of the article

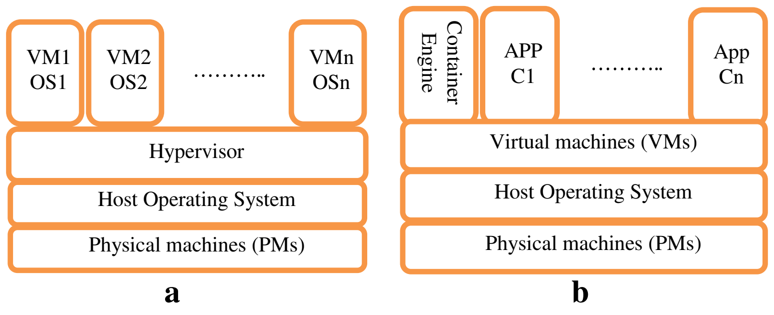

Fig. 1 Virtualization technology. **a**. Virtual machine architecture. **b**. Container virtualization architecture

To provide containers as services, a relatively new cloud service model based on container virtualization, called Container as a Service (CaaS), has emerged. CaaS solves the problem of applications developed in a certain PaaS environment and whose execution is restricted to that PaaS environment's specifications. Therefore, CaaS frees the application, making it completely independent of the PaaS specification environment, and eliminates dependencies. Hence, container-based applications are portable and can execute in any execution environment [4]. Several container services are currently available on public clouds, including Google Container Engine and Amazon EC2 Container Service (ECS).

Docker, which is an open platform for executing application containers [5], comprises an ecosystem for container-based tools. A Docker Swarm functions as a scheduler for the containers on VMs. However, the Docker Swarm placement algorithm simply maps containers onto the available resources in round-robin fashion [6], and the round-robin placement algorithm does not consider resource usage or the utilization of either PMs or VMs. The problem of container or task placement has been widely studied [7–9]. This problem is critical because it has a direct impact on the utilization of PMs, the performance of VMs, the number of active PMs, the number of instantiated VMs, energy consumption, and billing costs. This paper investigates the following research problem:

- **Container placement:** This involves mapping new containers to VMs by considering VMs placements on the PMs at the same time. The placement aims to minimize instantiated VMs, minimize active PMs and to optimize the utilization of both the active PMs and VMs in terms of number CPU cores and memory usage.

In this paper, we propose a placement architecture for containers on VMs. The proposed architecture aims to maximize the utilization of both VMs and PMs and to minimize the number of instantiated VMs and active PMs. To this end, we explore the Best Fit and Max Fit heuristics along with a fitness function that evaluates resource usage as scheduling algorithms. Furthermore, a meta-heuristic Ant Colony Optimization based on Best Fit (ACO-BF) in combination with the proposed fitness function is also proposed.

The scope of the study is as follows:

- Each job consists of a set of independent tasks, and each task executes inside a container.
- Each task is represented by a container whose required computing resources, CPU and memory, are known in advance. The CPU and memory resources are the only resources considered in the study.
- Each container executes inside a single VM.

The remainder of this paper is organized as follows. Section 2 discusses the related research background. The proposed placement architecture and algorithms are introduced Section 3. Section 4 discusses the experimental evaluation and presents the feasibility study of the proposed algorithms. Section 5 concludes the paper and presents future work.

Background and related work

Cloud computing is led by the virtualization technology. A virtual machine is an emulation of physical hardware resources that runs an independent operating system on top of a VMM, also called a hypervisor. The hypervisor acts as a management layer for the created virtual machines. Recently, however, a new virtualization technology has emerged, namely, containerization. A container is a lightweight virtual environment that shares the operating system kernel with other containers. Docker is an open-source application development platform in which developed applications can execute in either a virtual machine or on a physical machine [6]. Docker Containerization is considered lightweight for two main reasons: (1) it significantly decreases container startup times because the containers do not require launching a VM or an operating system, and (2) communications are performed through system calls using the shared operating system kernel—unlike the heavyweight communication performed by a virtual machine hypervisor.

To this end, a container is considered an isolated workload unburdened by the overhead of a hypervisor. An application's tasks execute inside the containers, and the containers are scheduled to virtual machines, which are mapped to physical machines. Consequently, it is challenging to design an effective placement algorithm that considers the utilization of both VMs and PMs.

Placement problems have been considered in the literature from the following different perspectives:

- Placement of virtual machines on physical machines
- Placement of containers on physical machines
- Placement of containers on virtual machines
- Placement of containers and virtual machines on physical machines.

Placement of VMS on PMs (VM-PM)

A virtual machine is a simulation of physical hardware resources, including processors, memory, and I/O devices. Each VM runs an independent operating system on top of the VMM. Previous studies on the problem of VM-PM placement were mainly intended to optimize the utilization of PMs in IaaS cloud infrastructure and to maintain certain Quality of Service (QoS) requirements of the applications submitted to IaaS clouds. Much of this research was conducted to investigate VM placement and provisioning on PMs, including static and dynamic scheduling, and optimal PM utilization was their main objective [9–11].

Placement of containers on PMs (C-PM)

Docker is an operating system-level virtualization mechanism that allows Linux containers to execute independently on any host in an isolated environment. For container placement, algorithms such as Docker Swarm and Google Kubernets [12, 13] have been proposed. The Docker Swarm allocation algorithm simply maps containers to the available resources in round-robin fashion [6]; however, many other scheduling algorithms have been proposed. The main goals of these scheduling algorithms are to either maximize the utilization of PMs or to maximize container performance. For example, in [14], an architecture called OptiCA was proposed in which each component of the distributed application is executed inside a container, and the containers are deployed on PMs. The main goal is to optimize the performance of big-data applications within the constraints of the available cores of the PMs. In [15], an algorithm was proposed to minimize the energy consumption of a data center by placing applications on the PMs efficiently under a certain defined service level agreement constraint while simultaneously minimizing the number of required PMs. A simulation was used to model the workload using a Poisson distribution with a queuing routing strategy to map the workload to the determined PMs.

Placement of containers on VMs (C-VM)

The placement algorithms of VMs on PMs are mainly intended to optimize PMs utilization. Currently, containerization technology has been used to optimize the utilization of VMs. Application containers are provided by Amazon EC2 in the form of a CaaS cloud service. In the CaaS model, tasks execute inside containers that are allocated to VMs to minimize the application response time and optimize the utilization of the VMs. For example, the proposed queuing algorithm in [15] can be used to map the containers directly onto VMs rather than onto PMs to minimize the required number of VMs and, consequently, to minimize energy consumption. In [16], a placement algorithm for containers on VMs was proposed. The main goal of the proposed placement algorithm was to reduce the number of instantiated VMs to reduce billing costs and power consumption. The proposed model used the constraint satisfaction programming (CSP) approach. In [17], a meta-heuristic placement algorithm was proposed that dynamically schedules containers on VMs based on an objective function that includes the number of PMs, PM utilization, SLA violations, migrations and energy consumption. In [18], the proposed placement algorithm was based on using Constraint Programing (CP) to map containers to VMs with the goal of minimizing the number of required VMs.

Two-tier placement of containers and VMs on PMs (C-VM-PM)

The two-tier placement algorithm schedules containers and VMs on PMs. The objectives of such placement algorithms are to (1) maximize the utilization of PMs and VMs, (2) minimize the number of PMs and VMs, and (3) improve the overall performance of the executed applications. This type of placement algorithm is the main concern of this research.

For example, in [19], a two-level placement algorithm is proposed for scheduling tasks and VMs on PMs simultaneously. The main goal of the proposed algorithm is to maximize PMs utilization. The placement algorithm first schedules VMs on the PMs using the Best Fit algorithm and then schedules tasks on the VMs using Best Fit. The results are compared with the those of FCFS scheduler. However, the proposed scheduler focuses on PMs utilization and does not simultaneously consider the utilization of both VMs and active PMs.

In [20], the Best Fit algorithm is used to load-balance the tasks over VMs by first considering the placement of VMs on active PMs using the Best Fit algorithm. This approach aims to ensure moderate PMs utilization. However, the focus of the study involves VM placement on PMs.

In [21], a placement algorithm is proposed based on a fitness function that considers the utilization of both the PMs and the VMs. The Best Fit placement algorithm is used along with the fitness function to evaluate the selections of both PMs and VMs. The experimental results show that the proposed algorithm improved PM utilization compared with the Best Fit algorithm without the fitness function. In this paper, a similar fitness function is proposed based on the remaining available resource percentages of both VMs and PMs. The fitness function is employed in the placement algorithms. Furthermore, in this paper, we explore using different heuristics, such as Best Fit, Max Fit and the meta-heuristic Ant Colony Optimization (ACO) algorithm.

The proposed (C-VM-PM) placement architecture and scheduling algorithms

As mentioned above, most existing research has focused on the problem of scheduling containers on either VMs or PMs or on the placement of VMs on PMs. Thus, the existing provision architectures and container-scheduling algorithms for VMs do not consider the VMs and PM attributes together. This leads to major fragmentation of the PM's resources and poor PMs utilization. For example, consider the placement scenario for container C5 using the configuration shown in Fig. 2. There are several placement options, and each placement decision impacts the utilization of VMs and PMs. The first placement option is the placement of container C5 in either VM1 or VM2. This placement improves the utilization of VM1 or VM2. The second placement option is the placement of container C5 in VM4. This placement option improves the utilization of PM2 as well as the utilizations of VMs on that PM. Finally, the last placement option is the placement of container C5 in VM4. This placement option is the worst option where an unnecessary PM is switched on. Without this placement option, PM3 can be put to sleep to minimize the number of PMs used.

This section presents the proposed placement architecture and scheduling algorithms proposed to improve PM and VM utilization while scheduling containers in a CaaS cloud service model. The architecture aims to minimize the number of instantiated VMs and active PMs and maximize their resource utilization.

The proposed placement architecture

In CaaS, tasks execute inside containers that are scheduled on VMs, which, in turn, are hosted on PMs. The objective of the placement architecture is to improve the overall resources utilization of both VMs and PMs, and to minimize the number of instantiated VMs and the number of active PMs. An overall view of the proposed architecture is shown in Fig. 3. The architecture consists of the following components:

Physical machine controller (PMC)

The PMC is a daemon that runs on each PM. The PMC collects information such as the utilization of each PM and its resource capacity. The PMC sends the collected data to the container scheduler.

VM controller (VMC)

A VMC is a daemon that runs on each instantiated VM. The VMC collects information such as the resource capacity of the VMs and its used resources. The VMC sends

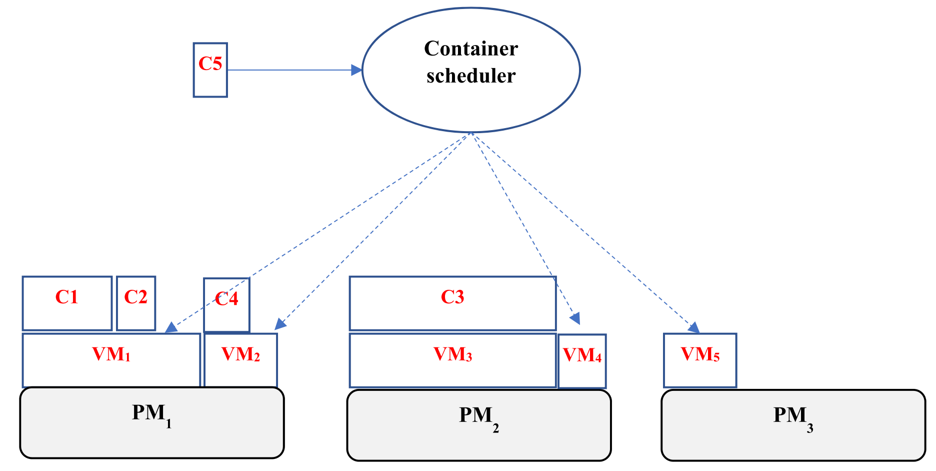

Fig. 2 An example of a container placement

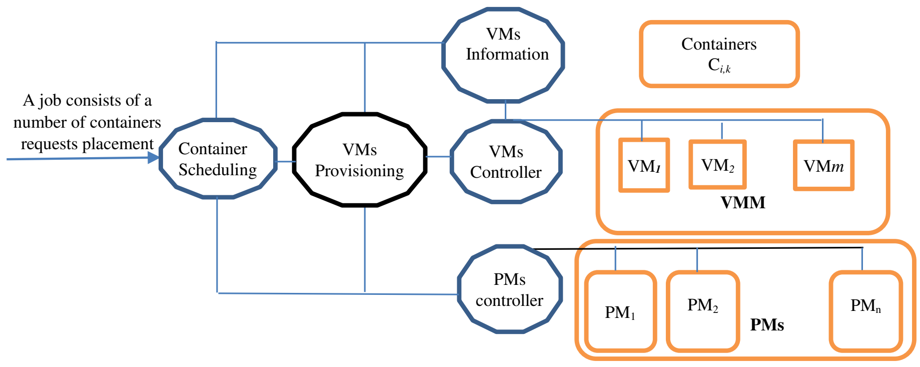

Fig. 3 The proposed architecture

the collected data to the container scheduler and to the VM information repository.

VM information repository (VMIR)

The VMIR maintains information about the types of VMs that can be instantiated and information about each instantiated VM.

VM provisioning (VMP)

The VMP schedules the placement of newly instantiated VMs on available PMs using the information available from the VMIR and PMC. Furthermore, the VMP is responsible for VMs provisioning.

Container scheduler (CS)

The CS determines the placements of containers on the VMs. The placement decisions are based on the information available from the PMC and the VMIR.

In the proposed architecture, each job consists of a number of independent tasks, each of which executes inside a container. The Container Scheduler (CS) contacts the VMIR to obtain the available resource capacity of each VM. The CS applies the placement algorithms that are proposed in the following subsections to find a placement for each container on a VM. When the CS finds a placement, it contacts the VMC to launch the container. When there is no VM available with enough physical resources to host the container, a new VM must

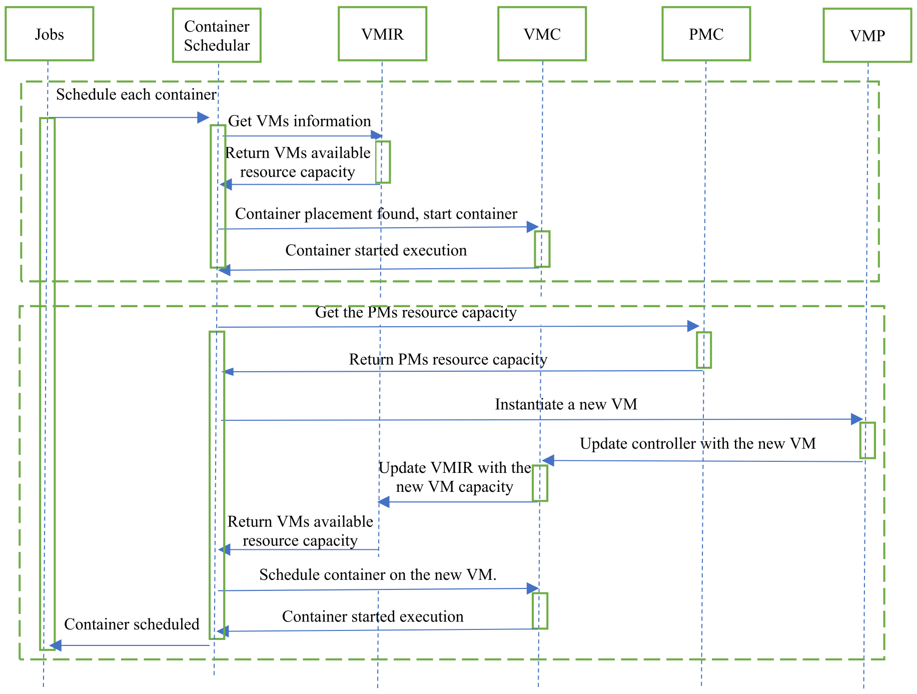
Jobs
Container Schedular
VMIR
VMC
PMC
VMP
Schedule each container
Get VMs information
Return VMs available resource capacity
Container placement found, start container
Container started execution
Get the PMs resource capacity
Return PMs resource capacity
Instantiate a new VM
Update controller with the new VM
Update VMIR with the new VM capacity
Return VMs available resource capacity
Schedule container on the new VM.
Container started execution
Container scheduled

Fig. 4 Sequence diagram for scheduling a container in the proposed architecture

Table 1 Symbol and descriptions

Symbol	Description
vm_{ij}	Virtual machine *i* on physical machine *j*.
pm_j	Physical machine *j*.
$C_{(k,i,j)}$	Container number *k* on vm_{ij}
$CC_{(k,i,j,r)}$	The container *k* capacity of resource *r* on vm_{ij}
$vmC_{(i,j,r)}$	The resource *r* capacity of vm_{ij}
$SC_{(j,r)}$	The resource *r* capacity of pm_j
$NC_{vm_{i,j}}$	Number of containers on vm_{ij}
Nvm_{pm_j}	Number of VMs on pm_j
N_{pm}	Total number of PMs
$CPU_{required}$	The required number of CPU cores for new container placement
$memory_{required}$	The required amount of memory for new container placement

be instantiated—either on a new or currently active PM. To accomplish this task, the CS asks the VMP to instantiate a new VM. The VMP finds a placement for the new VM using the information provided by the VMIR and the PMC. The VMP instantiates the new VM and updates the VMIR. Finally, the VMC starts the container on the newly instantiated VM. Figure 4 shows the sequence diagram for container placement in the proposed architecture.

The following subsection presents a mathematical formulation for the problem of container placement on VMs and PMs. The main objective is to improve the overall utilization of VMs and PMs. Based on the proposed formulation, heuristic and meta-heuristic scheduling algorithms will be presented.

Problem formulation

This subsection presents the mathematical formulation of the objective function for the proposed architecture, which minimizes the number of active VMs and PMs and maximizes the utilization of both VMs and PMs. Table 1 summarizes the symbols used in this section.

The goal of the objective function is to minimize the number of physical machines:

$$minimize\ N_{pm} = \sum_j Nvm_{pm_j}$$

We consider the following constraints on the scheduling decisions:

$$C_{(k,i,j)} = f(x) = \begin{cases} 1, container\ k\ is\ scheduled\ to\ vm_{i,j} \\ 0, otherwise \end{cases} \tag{1}$$

$$vm_{ij} = \begin{cases} 1, vm_i\ is\ scheduled\ on\ pm_j \\ 0, otherwise \end{cases} \tag{2}$$

$$\sum_j \sum_i C_{(k,i,j)} = 1 \forall k \tag{3}$$

$$\sum_i vmC_{(i,j,r)} \le SC_{(j,r)} \forall r = \{ cpu\_cores, memory, bandwidth, storage\}, j \tag{4}$$

$$\sum_i \sum_k CC_{(k,i,j,r)} \le SC_{(j,r)} \forall resource\ r, j. \tag{5}$$

Equation (4) ensures that the sum of each resource type allocated by all VMs on each PM is less than or equal to the total amount of that resource available on the PM.

Equation (5) ensures that the sum of each resource requested by the containers on all VMs vm_{ij} on the physical machine pm_j is less than or equal the total capacity $SC_{(j,r)}$ of that resource on that physical machine pm_j. This condition applies to computational resources, such as CPU cores, memory, storage, and bandwidth. The placement algorithms should maximize the resource utilization on

```
Algorithm: Best Fit for Container-VM-PM scheduling

Input: $C_k \in C$ set of Containers, a set of virtual machines $v_{ij}$, and a set of physical machines $pm_j$.
       $nContainer$ is the number of containers.
Output: Placement of $C_k$ on subset of $vm_{ij}$.

1. for $C_k$ from 1 to $nContainer$ do
2.     Find $vm_{ij}$ that has the minimum resource waste of both VM and PM using Equation (9)
3.     Place container $C_k$ on $vm_{ij}$.
4. End-for
5. For each container $C_k$ not allocated to VM do
6.     Instantiate a new $vm_{ij}$ from the list $pm_j$ that has the minimum resource waste using Equation (6)
7.     Place container $C_k$ on the $vm_{ij}$
8. End-for
```

Fig. 5 The Best Fit algorithm for container-VM-PM placement

```
Algorithm: Max Fit for Container−VM−PM scheduling

Input: $C_k \epsilon C$ set of Containers, a set of virtual machines $v_{ij}$, and a set of physical machines $pm_j$.
       $nContainer$ is the number of containers.
Output: Placement of $C_k$ on subset of $vm_{ij}$.

1. for $C_k$ from 1 to $nContainer$ do
2.    Find $vm_{ij}$ that has the max resource waste of both VM and PM using Equation (9)
3.    Place container $C_k$ on $vm_{ij}$.
4. End-for
5. For each container $C_k$ not allocated to VM do
6.    Instantiate a new $vm_{ij}$ from the list $pm_j$ that has the maximum resource waste using Equation (6)
7.    Place container $C_k$ on the $vm_{ij}$
8. End-for
```

Fig. 6 The Max Fit algorithm for container-VM-PM placement

the VMs and PMs. This can be achieved by minimizing the resource waste on both the VMs and PMs simultaneously. Equation (6) calculates the normalized resource waste on a PM considering only the number of CPU cores and the memory resources.

$$PF(j) = w_1 \times \frac{PM\_available_j^{CPU\_cores}}{SC_{(j,r=cpu\ cores)}} + w_2 \times \frac{PM\_available_j^{memory}}{SC_{(j,r=memory)}} \tag{6}$$

w_1 and w_2 are weights constants. $PM\_available_j^{CPU\_cores}$ and $PM\_available_j^{memory}$ represent the available number of cores and the amount of memory on pm_j after assigning the requested physical resources, respectively, and they are calculated using Eqs. (7, 8).

$$PM\_available_j^{CPUcores} = SC_{(j,r=cpu\_cores)} - \left(\sum_i \left(vm_{ij} \times \sum_k CC_{(k,i,j,r=cpu\_cores)} \right) + CPU_{required}\right) \tag{7}$$

$$PM\_available_j^{memory} = SC_{(j,r=memory)} - \left(\sum_i \left(vm_{ij} \times \sum_k CC_{(k,i,j,r=memory)} \right) + memory_{required}\right) \tag{8}$$

```
Algorithm: ACO-BF for container-VM-PM scheduling

Input: $C_k \epsilon C$ set of Containers, a set of virtual machines $v_{ij}$, and a set of physical machines $pm_j$
Output: Placement of $C_k$ on subset of $vm_{ij}$.

1.  Initialize the parameters {$Max\_Iteration$, $nAnts$ number of ants, $Q, \alpha, \beta, \rho$}
2.  If (there are no existing $vm_{ij}$ with sufficient resources to accommodate the resources required by $C$)
       Instantiate set $vm_{ij}$ for C on subset $pm$ that maximize the utilization using Best Fit and Equations (6and9).
3.  End-If
4.  Place all ants randomly for placement of the set containers $C_k$ on $vm_{ij}$ and $pm_j$.
5.  Initialize the initial pheromone trail $\tau_{ij}$ using Equation (13)
6.  Initialize the placement cost $\eta_{ij}$ using Equation (15)
7.  for $iter$ from 1 to $Max\_Iteration$ do
8.     for $ant_k$ from 1 to $nAnts$ do
9.        for $C_k$ from 1 to $nContainer$ do
10.          Calculate the probability function $P_{ij}^k(t)$ for placement of $C_k$ on each $vm_{ij}$ and $pm_j$ using Equation (14)
11.          $ant_k$ chooses $vm_{ij}$ for $C_k$ according to the Roulette method.
12.          Add the selected $vm_{ij}$ to the taboo table as a placement of $C_k$ for $ant_k$.
13.       End-for
14.       Calculate the cost of placement containers $C$ for $ant_k$
15.    End-for
16.    Update the pheromone trail $\tau_{ij}$ using Equation (12)
17. End-for Repeat until the maximum number of iterations is reached or the best placement is found
```

Fig. 7 The ACO algorithm for container-VM-PM placement

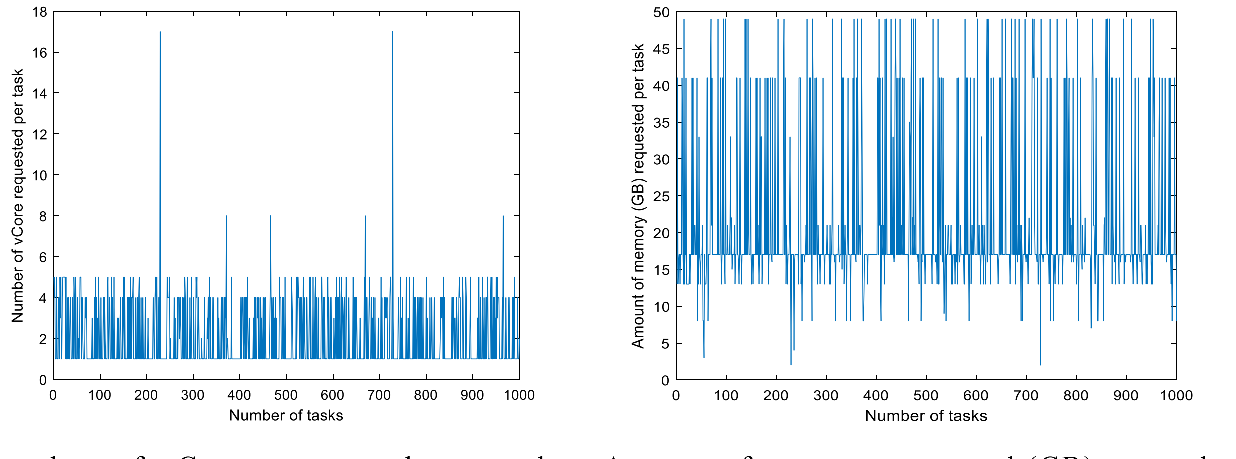

Fig. 8 A sample of the vCore and memory requests for 1000 tasks

$$VF(i,j) = w_1 \times \frac{VM\_available_{ij}^{CPU\_cores}}{vmC_{(i,j,r=CPU\_cores)}} + w_2 \times \frac{VM\_available_{ij}^{memory}}{vmC_{(i,j,r=memory)}} + PF(j) \times vm_{ij} \quad (9)$$

Equation (9) calculates the normalized resource waste of a VM while considering the resource waste of the PM hosting that VM. $VM\_available_{ij}^{CPU\_cores}$ and $VM\_available_{ij}^{memory}$ represent the number of cores available and the amount of memory on vm_{ij} after assigning the requested virtual resources, respectively, and they are calculated using Eqs. (10) and (11).

$$VM\_available_{ij}^{CPUcores} = vmC_{(i,j,r=CPU\_cores)} - \left(\sum_k \left(C_{(k,i,j)} * CC_{(k,i,j,r=CPU\_cores)}\right) + CPU_{required}\right) \quad (10)$$

$$VM\_available_{ij}^{memory} = vmC_{(i,j,r=memory)} - \left(\sum_k \left(C_{(k,i,j)} * CC_{(k,i,j,r=memory)}\right) + memory_{required}\right) \quad (11)$$

The placement algorithm should make its decisions based on the minimum values of Eqs. (6) and (9).

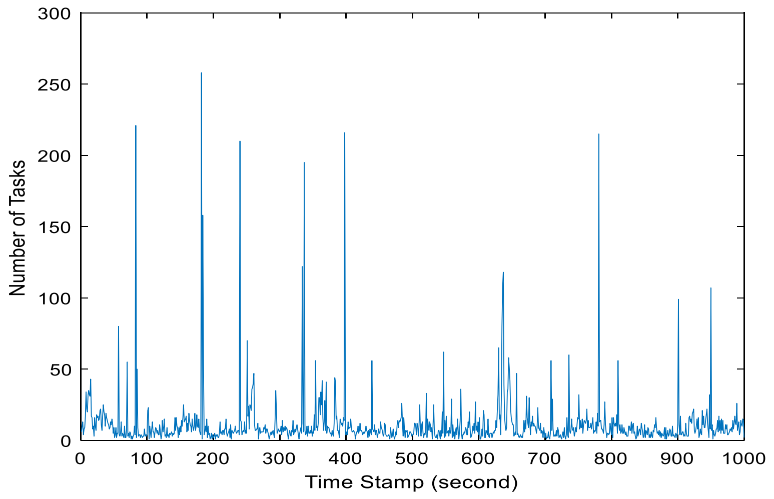

Fig. 9 The number of tasks to be scheduled during a 1000-s period

However, this problem is NP-hard and the difficulty increases exponentially as the number of containers increases [9]. The following subsection discusses the heuristic and meta-heuristic scheduling algorithms.

The proposed scheduling algorithms

This subsection discusses the proposed scheduling algorithms for scheduling containers on VMs and scheduling VMs on PMs. The problem can be viewed as a multitier bin-backing problem in which the VMs are considered the bins for the containers, while simultaneously, the PMs are considered the bins for the VMs. The goals of the proposed scheduling algorithms are to (1) maximize PMs utilization and (2) maximize VMs utilization. To achieve these goals, we propose three algorithms using the mathematical formulation discussed in Section 3.2. The algorithms are Best Fit, Max Fit, and a meta-heuristic based on Ant Colony Optimization and Best Fit.

Best fit algorithm

The Best Fit algorithm (BF) is a heuristic that is used to optimize the bin packing problem. The BF algorithm is widely used in the literature to find placements for VMs on PMs [22]. In this study, the BF algorithm is used to find a placement for containers on VMs by considering the resource utilizations of both the VMs and the PMs. The BF algorithm presented in Fig. 5 schedules containers onto VMs that have the least available resource capacity that can accommodate the resources required by the new container. This is achieved by scheduling the container to the VM with the minimum resource wastage, using Eq. (9). When no existing VM has sufficient available resources to accommodate the resources required by the new container, a new VM must be instantiated.

The same idea is applied to the placement of VMs onto PMs: a new VM is instantiated on the PM with the least available resource capacity that can accommodate the resources required by the new VM using Eq. (6). Finally, when there is no PM available whose available resource capacity fits the resources required by the new VM, a new PM is activated. The Best Fit algorithm is shown in Fig. 5.

Max fit algorithm

The Max Fit (MF) algorithm is similar to the Best Fit heuristic. However, the main difference is that Max Fit elects to place containers on VMs that have the maximum available resource capacity and can accommodate the resources required by the new container. Thus, in Max Fit, containers are scheduled to VMs that have the maximum resource wastage using Eq. (9). When no VM has sufficient available resources to accommodate the resources required by the new container, a new VM must be instantiated. The new VM is instantiated on the active PM with the maximum available resource capacity that can accommodate the resources required by the new VM using Eq. (6). Finally, when no PM exists whose available resource capacity can accommodate the resources required by the new VM, a new PM is activated. The Max Fit algorithm is shown in Fig. 6.

Table 2 Virtual machine types of instances

VM Type	vCPU	Memory Size (GB)	VM Type	vCPU	Memory Size (GB)
Type 1	1	1	Type 9	16	64
Type 2	1	2	Type 10	16	128
Type 3	2	4	Type 11	32	128
Type 4	2	8	Type 12	32	256
Type 5	4	8	Type 13	48	256
Type 6	4	16	Type 14	48	256
Type 7	8	32	Type 15	64	1024
Type 8	8	64	Type 16	64	2048

Ant colony optimization (ACO)

ACO is a meta-heuristic inspired by real ant colonies and their collective foraging behavior for food sources. ACO is a probabilistic method used to solve discrete optimization problems, such as finding the shortest path in a graph [23]. In the search for food, an ant leaves the nest and travels along a random path to explore the surrounding area for food. During the search, the ant produces pheromone trails along its path. Other ants smell the pheromones and choose the path with a higher pheromone concentration. During the ant's return trip, it releases a quantity of pheromone trails proportional to the quantity and quality of the food it found. All the remaining ants of the colony will follow paths with higher pheromone concentrations at a higher probability than they follow paths with lower pheromone concentrations. Eventually, all the ants will follow a single optimal path for food. For m ants and n possible paths, each ant determines its path according to the concentration of pheromone trail in each path of the possible paths. Initially, all the ants select a path randomly because of the negligible differences in pheromone quantity among the paths [24, 25].

Table 3 Physical machine configuration attributes

Server Type	Number CPUs	Number of Cores/CPU	Memory Size (GB)
Type 1	32	4	1024
Type 2	64	8	2048

Table 4 Parameter settings of the conducted experiments

Parameter	Description	Value
nContainer	Number of containers request scheduling	25–250
Max_Iteration	Maximum number of iterations in ACO	100
nAnts	Number of ants	8
α	Heuristic constant	0.1
β	Heuristic constant	2
Q	Heuristic constant	1
ρ	Evaporation rate	0.2
w_1	First resource normalization weight	0.5
w_2	second resource normalization weight	0.5

The ACO algorithm has proven effective as an optimization algorithm for a number of problems, including the traveling salesman and job shop scheduling [26] problems. Furthermore, the ACO algorithm has been used to efficiently schedule VMs on cloud resources with the objective of minimizing total resource wastage and power consumption [27]. The ACO has also been used to schedule tasks on cloud VMs with the goal of load-balancing the tasks on the VMs and reducing the make span of the tasks [28]. A number of schedulers have adopted the ACO algorithm to load balance tasks on cloud resources [29, 30]. The following proposed algorithm uses ACO to find container placements on VMs with the objective of maximizing the utilization of VMs and PMs while minimizing the number of instantiated VMs and active PMs.

In a search for container placement on vm_{ij} and pm_j with the objective of maximizing the used PMs and VMs utilization, the following equation represents the pheromone trail of that container placement for an ant k in iteration $(t + 1)$.

$$\tau_{ij}^k(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t), \tag{12}$$

where $\Delta\tau_{ij}^k$ represents the amount of pheromone released by an ant due to the quality of the placement of a container on vm_{ij} and pm_j. Here, ρ is a constant that represents the rate of pheromone evaporation to simulate the effect of the evaporation of the pheromone at each step, and $\Delta\tau_{ij}^k$ is calculated using the following equation:

$$\Delta\tau_{ij}^k = \frac{1}{VF(i,j)}, VF(i,j) \textit{ is calculated using Equation } (9) \tag{13}$$

$VF(i,j)$ is the fitness of the placement on vm_{ij} and represents the probability that an ant k will select the placement of a container on vm_{ij} and pm_j.

$$P_{ij}^k(t) = \frac{(\tau_{ij}(t))^\alpha \left(\eta_{ij}(t)\right)^\beta}{\sum_s (\tau_{is}(t))^\alpha \left(\eta_{is}(t)\right)^\beta}, \alpha, \beta \textit{ are heuristic constants}, \tag{14}$$

where $\eta_{ij}(t)$ is a heuristic function that represents the quality of a container placement, and $\eta_{is}(t)$ is calculated as follows:

$$\eta_{ij}(t) = \frac{Q}{VF(i,j)}, Q \textit{ is a heuristic constant}, \tag{15}$$

The complete algorithm is shown in Fig. 7. The proposed algorithm starts by calculating the total amount of resources available in the instantiated set vm_{ij}. If the available resources are below the total amount of resources required by the set of containers C, a new subset of VMs is instantiated using the BF algorithm that satisfies Eqs. (6) and (9), which maximize the VMs and PMs utilization. The algorithm starts by calculating

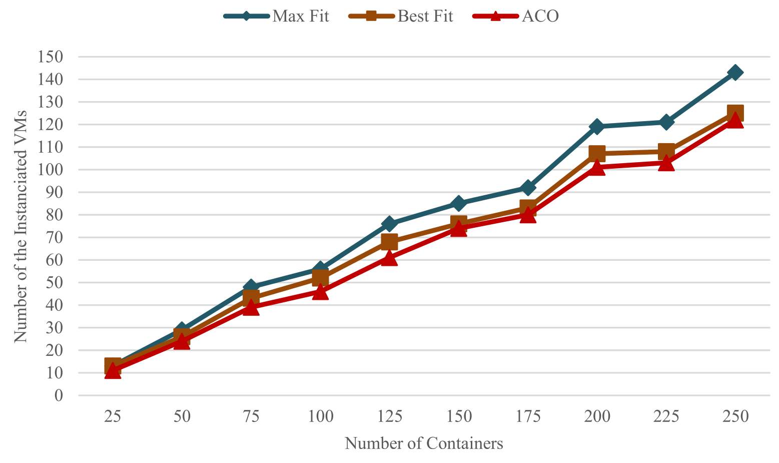

Fig. 10 Number of instantiated VMs

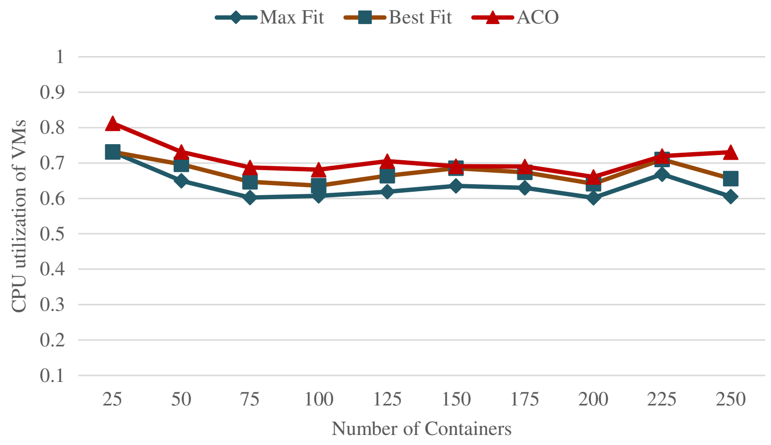

Fig. 11 CPU utilization of instantiated VMs

the initial pheromone trail matrix τ_{ij} and the placement cost matrix η_{ij} using Eqs. (13) and (15), respectively. During each iteration, each ant ant_k finds a placement for the set of containers C_k on vm_{ij} with a probability calculated using Eq. (14). The selection of vm_{ij} is performed using the roulette wheel method where a random number ϵ [0, 1] is generated and a cumulative sum of the probabilities is calculated. The cumulative sum probabilities are arranged in ascending order, and the cumulative probability equal to or greater than the generated random number is selected. After each iteration, the ACO algorithm updates the pheromone trail matrix τ_{ij}. The algorithm iterates until the maximum number of iterations is reached or the best placement of the containers on the subset of VMs is found.

Performance evaluation

This section presents the experiments conducted to evaluate the proposed placement architecture and scheduling algorithms. A program is designed and developed using MATLAB to simulate the proposed placement algorithms. A simulation-based evaluation is adopted because it allows the environment parameters to be controlled and the experiments to be repeated under different constraints. The experiments are conducted to evaluate the proposed scheduling algorithms in terms of the utilization of VMs and PMs. The following subsections discuss the workload utilized for the simulation, the parameter settings and the evaluation of the proposed algorithms for effective placement of container-based cloud environments.

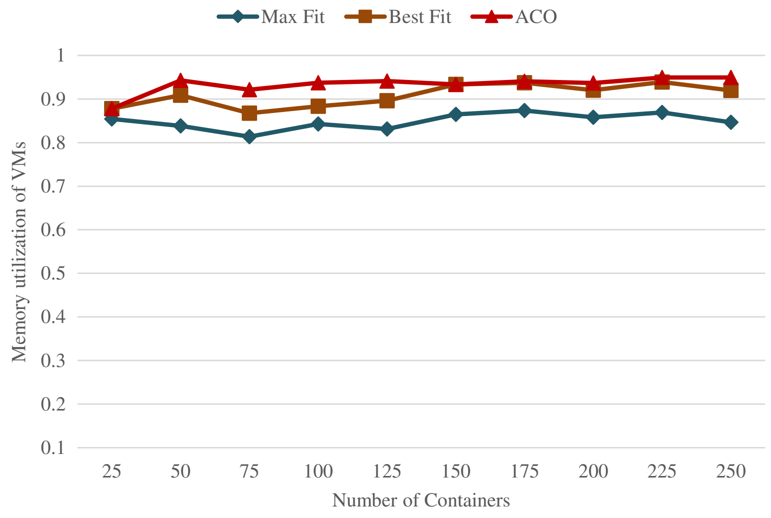

Fig. 12 Memory utilization of instantiated VMs

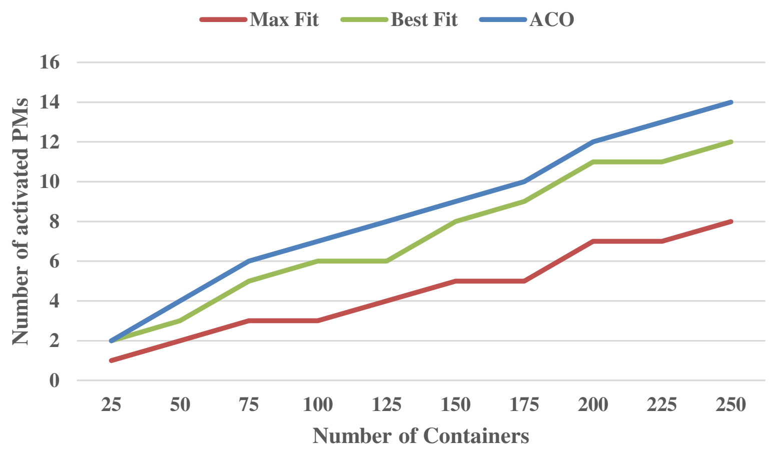

Fig. 13 Number of activated PMs

Google cloud workload

A workload is a set of jobs submitted for allocation and execution in a cloud environment. The workload dataset used in the evaluation is the Google Cloud trace log, which was published in 2011 [31]. The Google real workload dataset is used because it is widely used in the literature for the purpose of workload placement and scheduling evaluations. In this paper, the workload is used to evaluate the applicability of the proposed placement algorithms in real world scenarios. The workload consists of a number of tables that describe the submitted task requirements, machine configurations, and task resource usage. The Task Event table contains information such as time, job ID, and task ID. Additionally, the table includes normalized data, such as resource requests for CPU cores and RAM. The number of vCores used to denormalize the CPU requests in the task Event table is 64. The amount of memory used to denormalize memory requests in the task Event table is 1024 GB. Figure 8 shows a sample of the vCore and memory requests for 1000 tasks in the Google cloud workload dataset, while Fig. 9 shows the number of tasks received for scheduling during a 1000-s period in the sample Google workload. Each task in the workload, the task Event table, is used to represent a container that requires a placement using the available VMs and PMs configurations described in the following subsections. The required CPU cores and memory request for each task, after denormalization, are considered the required resources for container placement.

Virtual machine configuration

The VM type repository includes several types of VMs that can be instantiated. Each VM type has distinct resource characteristics, including the number of vCPUs, the memory size, and the storage size. Each vCPU in a VM is allocated one core of the PM. Table 2 presents the different VM types used in the experiment.

PM configuration

Table 3 shows simulated configurations of the physical machines used in the evaluation experiment.

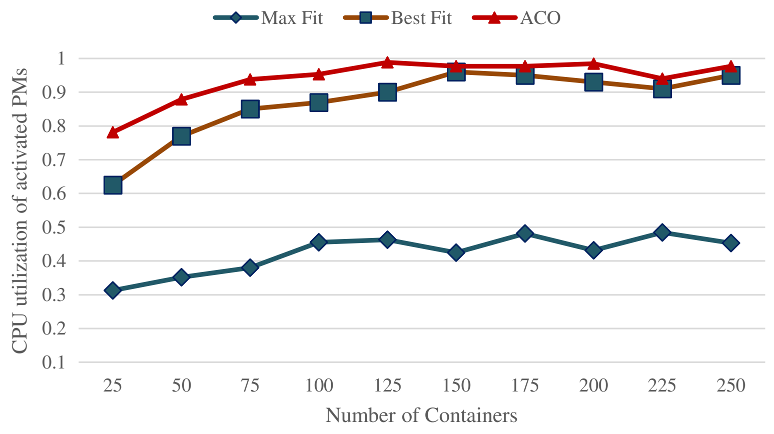

Fig. 14 CPU utilization of activated PMs

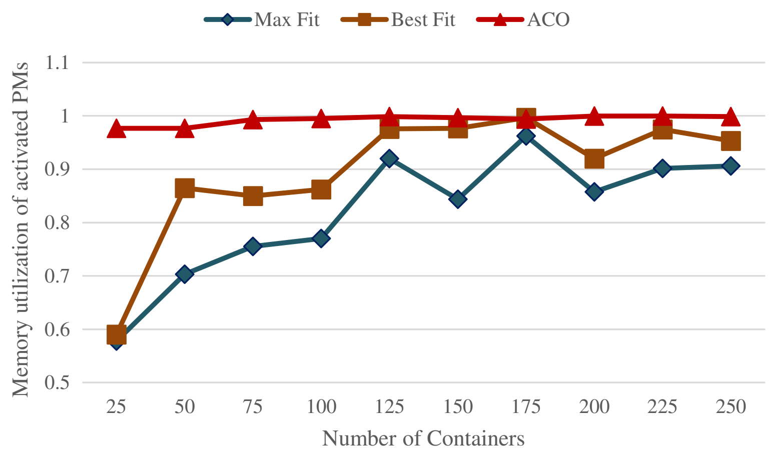

Fig. 15 Memory utilization of activated PMs

Parameters setting

The simulation was run on a PC with an Intel Core i7–4510 CPU at 2.60 GHz and 8 GB RAM. In the experiments the parameters are set to the values described in Table 4. The parameter settings are obtained by conducting several preliminary experiments.

Experiments and evaluation

The goal of the experiments was to evaluate the proposed placement architecture using the Best Fit, Max Fit, and ACO-BF scheduling algorithms. The placement architecture works as a multitier scheduling algorithm in which the placement decision considers the utilization of both PMs and VMs. Each scheduling algorithm is executed 10 times for different number of containers. Initially, all PMs are off with no instantiated VMs. The output of each algorithms is the set of VMs to instantiate with their corresponding PMs to turn on.

Figure 10 shows the number of instantiated VMs for different numbers of containers. It is observed that the ACO-BF scheduler guarantees the lowest number of instantiated VMs compared to the other schedulers. Furthermore, the Max Fit algorithm instantiates a larger number of VMs.

Figures 11 and 12 show the average CPU and memory utilization of the instantiated VMs using the proposed scheduling algorithms for different numbers of containers. It is clear that the proposed ACO-BF algorithm yields the best resource utilization of VMs.

Figure 13 shows the number of PMs turned on for scheduling different number of containers. It is observed that the proposed ACO-BF algorithm turns on a larger number of PMs than the Best Fit and Max Fit algorithms. The Max Fit algorithms activates the lowest number of PMs.

Figures 14 and 15 show the CPU and memory utilization of the activated PMs for different numbers of containers. The proposed ACO-BF placement algorithm achieves the highest resource utilization. It is clear that the Max Fit algorithm guarantees a small number of active PMs but achieves poor resources utilization, as it activates the PMs with largest the physical resources. The proposed ACO-BF placement algorithm activates a large number of PMs because it turns on the PMs with the lowest resource capacities that can accommodate the resources required by the scheduled containers.

Figure 16 shows the time costs of the heuristics, which are computationally lightweight compared to the

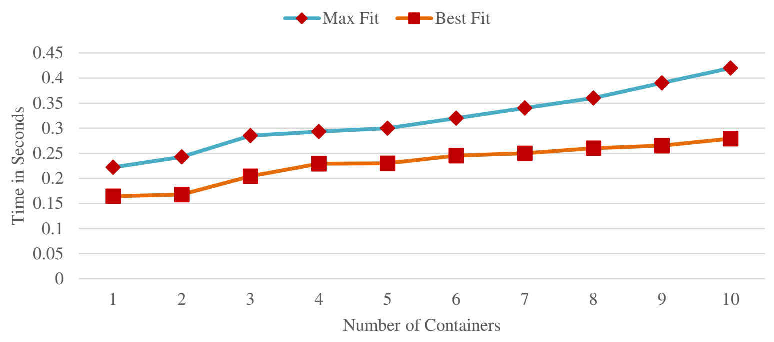

Fig. 16 The time cost of the scheduling heuristics

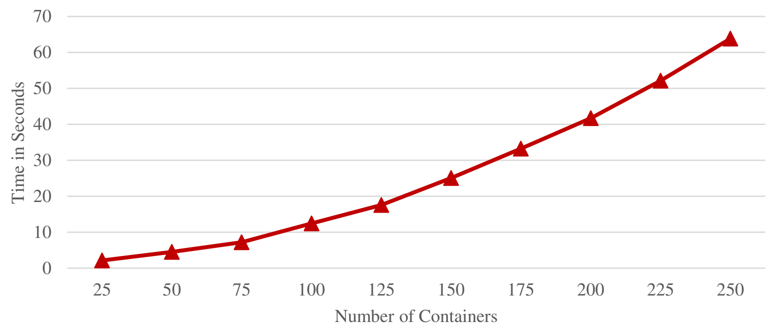

Fig. 17 The time cost of the proposed metaheuristic ACO

meta-heuristic ACO-BF, as shown in Fig. 17. Although the ACO requires a substantial amount of time compared to the heuristic algorithms, it guarantees the highest utilization of both VMs and PMs.

Conclusions and future work

In this paper, a placement architecture is presented for container-based tasks on a cloud environment. The proposed architecture is a multitier scheduler that aims to maximize the utilization of both the active PMs and the instantiated VMs while minimizing the number of active VMs and PMs. Two scheduling algorithms, including the heuristic Max Fit and Best Fit algorithm as well as the meta-heuristic ACO, are tested using a fitness function that analyzes the percentage of remaining resources that are wasted in the PMs and VMs. The evaluation shows that the ACO is highly effective in terms of VM and PM utilization and in minimizing the number of active PMs and instantiated VMs. Furthermore, but the proposed ACO increases the number of active PMs because it activates PMs with the lowest configurations that can accommodate the required resources.

This study assumed that each job consists of independent containers. In the future, we will study workflow containerization processes in which each workflow consists of a set of dependent communicating containerized tasks. This type of workflow is represented as a Directed Acyclic Graph (DAG), where each containerized task requires a specific execution environment to satisfy the priorities and communication constraints of the workflow. Furthermore, this study considered only the number of CPU cores and the memory size as computing resources in the proposed scheduling algorithms. However, in workflow scheduling, additional computing resources that must be considered include processing cores, memory, storage, and data transfers. These additional considerations arise because PMs are distributed; thus, data transfer bandwidth is an important factor that must be considered in the workflow scheduling problem. Furthermore, maintaining the quality of service (QoS) of the workflows submitted by the users of the executed containers in terms of SLA is another important research issue. Finally, additional meta-heuristics such as Particle Swarm and Cuckoo Search, should be tested for scheduling and provisioning workflow containers on cloud environments.

Abbreviations

ACO: Ant Colony Optimization; BF: Best Fit; CaaS: Container as a Service; MF: Max Fit; PMs: Physical Machines; VMs: Virtual Machines

Acknowledgements
Not applicable.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Authors' contributions
All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Faculty of Computers and Informatics, Suez Canal University, Ismailia, Egypt. [2]Faculty of Computers and Information Technology, University of Jeddah, Jeddah, Kingdom of Saudi Arabia.

Received: 11 January 2019 Accepted: 13 May 2019
Published online: 29 May 2019

References
1. Varghese B, Buyya R (2018) Next generation cloud computing: new trends and research directions. Future Gener Comput Syst 79:849–861
2. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25:599–616
3. Mell PM, Grance T (2011) SP 800–145. The NIST definition of cloud computing. National Institute of Standards & Technology, Gaithersburg, MD
4. Pahl C, Brogi A, Soldani J, Jamshidi P (2018) Cloud container technologies: a state-of-the-art review. IEEE Transactions on Cloud Computing. https://doi.org/10.1109/TCC.2017.2702586
5. What is Docker? Available from: https://www.docker.com/what-docker

6. Tao Y, Wang X, Xu X, Chen Y (2017) Dynamic resource allocation algorithm for container-based service computing. In: IEEE 13th international symposium on autonomous decentralized system (ISADS), IEEE, Bangkok, Thailand, 22–24 March 2017
7. Masdari M, Nabavi SS, Ahmadi V (2016) An overview of virtual machine placement schemes in cloud computing. J Netw Comput Appl 66:106–127
8. Pires FL, Barán B (2015) A virtual machine placement taxonomy. In: 15th IEEE/ACM international symposium on cluster, cloud and grid computing, IEEE, Shenzhen, China, 4-7 May 2015
9. Usmani Z, Singh S (2016) A survey of virtual machine placement techniques in a cloud data center. Procedia Comput Sci 78:491–498
10. Liu L, Zhe Q (2016) A survey on virtual machine scheduling in cloud computing. In: 2nd IEEE international conference on computer and communications (ICCC), IEEE, Chengdu, China, 14-17 October 2016
11. Eswaraprasad R, Raja L (2017) A review of virtual machine (VM) resource scheduling algorithms in cloud computing environment. Journal of Statistics and Management Systems 20:703–711
12. Dziurzanski P, Indrusiak LS (2018) Value-based allocation of docker containers. In: 26th Euromicro international conference on parallel, distributed and network-based processing (PDP), IEEE, Cambridge, UK, 21-23 March 2018
13. Jaison A, Kavitha N, Janardhanan PS (2016) Docker for optimization of cassandra NoSQL deployments on node limited clusters. In: International conference on emerging technological trends (ICETT), IEEE, Kollam, India, 21–22 October 2016
14. Spicuglia S, Chen LY, Birke R, Binder W (2015) optimizing capacity allocation for big data applications in cloud datacenters. In: IFIP/IEEE international symposium on integrated network management (IM), IEEE, Ottawa, ON, Canada, 11–15 May 2015
15. Raj VKM, Shriram R (2011) Power aware provisioning in cloud computing environment. In: International conference on computer, communication and electrical technology (ICCCET), IEEE, Tamilnadu, India, 18-19 March 2011
16. Tchana A, Palma ND, Safieddine I, Hagimont D, Diot B, Vuillerme N (2015) Software consolidation as an efficient energy and cost saving solution for a SaaS/PaaS cloud model. In: European conference on parallel processing, Springer, Berlin, Heidelberg, 25 July 2015
17. Yaqub E, Yahyapour R, Wieder P, Jehangiri AI, Lu K, Kotsokalis C (2014) Metaheuristics-based planning and optimization for SLA-aware resource management in PaaS clouds. In: IEEE/ACM 7th international conference on utility and cloud computing, IEEE, London, UK, 8–11 December 2014
18. Dhyani K, Gualandi S, Cremonesi P (2010) A constraint programming approach for the service consolidation problem. In: International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming, Springer, Berlin, Heidelberg, 14-18 June 2010
19. Dhanashri B, Divya T (2017) Implementation of two level scheduler in cloud computing environment. Int J Comput Appl 166:1–4
20. Innocent FM, Alphonsus M, Nansel L, Titus EF, Dashe A (2018) Best-fit virtual machine placement algorithm for load balancing in a cloud computing environment. Int J Sci Eng Res 9:6
21. Zhang R, Zhong A-M, Dong B, Tian F, Li R (2018) Container-VM-PM architecture: a novel architecture for docker container placement. In: International conference on cloud computing, Springer International Publishing, Cham, 19 June 2018
22. Shi W, Hong B (2011) Towards profitable virtual machine placement in the data center. In: Fourth IEEE international conference on utility and cloud computing, IEEE, Victoria, NSW, Australia, 5–8 December 2011
23. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE Comput Intell Mag 1:28–39
24. Xue X, Cheng X, Xu B, Wang H, Jiang C (2010) The basic principle and application of ant colony optimization algorithm. In: International conference on artificial intelligence and education (ICAIE), IEEE, Hangzhou, China, 29–30 October 2010
25. Nayyar A, Singh R (2016) Ant colony optimization — computational swarm intelligence technique. In: 3rd international conference on computing for sustainable global development (INDIACom), IEEE, New Delhi, India, 16–18 March 2016
26. Zhang J, Hu X, Tan X, Zhong JH, Huang Q (2006) Implementation of an ant colony optimization technique for job shop scheduling problem. Trans Inst Meas Control 28:93–108
27. 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. J Comput Syst Sci 79:1230–1242
28. Li K, Xu G, Zhao G, Dong Y, Wang D (2011) Cloud task scheduling based on load balancing ant colony optimization. In: Sixth annual chinagrid conference, IEEE, Liaoning, China, 22–23 August 2011
29. Mishra R, Jaiswal A (2012) Ant colony optimization: a solution of load balancing in cloud. International Journal of Web & Semantic Technology 3:33–50
30. Nishant K, Sharma P, Krishna V, Gupta C, Singh KP, Nitin, Rastogi R (2012) Load balancing of nodes in cloud using ant colony optimization. In: UKSim 14th international conference on computer modelling and simulation, IEEE, Cambridge, UK, 28–30 March 2012
31. Reiss C, Wilkes J, Hellerstein JL (2011) Google cluster-usage traces: format+ schema. Technical Report, Google Inc., Mountain View, CA, USA