



Article

GWO-Based Simulated Annealing Approach for Load Balancing in Cloud for Hosting Container as a Service

Manoj Kumar Patra ¹, Sanjay Misra ^{2,*}, Bibhudatta Sahoo ¹ and Ashok Kumar Turuk ¹¹ Department of Computer Science and Engineering, National Institute of Technology, Rourkela 769 008, India² Department of Computer Science and Communication, Østfold University College, 1783 Halden, Norway

* Correspondence: sanjay.misra@hiof.no

Abstract: Container-based virtualization has gained significant popularity in recent years because of its simplicity in deployment and adaptability in terms of cloud resource provisioning. Containerization technology is the recent development in cloud computing systems that is more efficient, reliable, and has better overall performance than a traditional virtual machine (VM) based technology. Containerized clouds produce better performance by maximizing host-level resource utilization and using a load-balancing technique. To this end, this article concentrates on distributing the workload among all available servers evenly. In this paper, we propose a Grey Wolf Optimization (GWO) based Simulated Annealing approach to counter the problem of load balancing in the containerized cloud that also considers the deadline miss rate. We have compared our results with the Genetic and Particle Swarm Optimization algorithm and evaluated the proposed algorithms by considering the parameter load variation and makespan. Our experimental result shows that, in most cases, more than 97% of the tasks were meeting their deadline and the Grey Wolf Optimization Algorithm with Simulated Annealing (GWO-SA) performs better than all other approaches in terms of load variation and makespan.

Keywords: cloud computing; container; load balancing; task scheduling; optimization; Metaheuristic's Methods



Citation: Patra, M.K.; Misra, S.; Sahoo, B.; Turuk, A.K. GWO-Based Simulated Annealing Approach for Load Balancing in Cloud for Hosting Container as a Service. *Appl. Sci.* **2022**, *12*, 11115. <https://doi.org/10.3390/app12211115>

Academic Editor: Eui-Nam Huh

Received: 12 September 2022

Accepted: 26 October 2022

Published: 2 November 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Cloud computing is a way of sharing computer system resources with more than one user using different virtualization technologies [1]. It has gained substantial popularity because of its pay-as-you-go business model. A pay-as-you-go business model is one in which you will have to pay the price for the number of resources or services you are using before you use it [2]. It helps lower service costs, and users can scale the resources as their business requirements change. The user needs to register with a cloud service provider (CSP) and request it over the internet to avail any service [3]. The cloud service provider has to satisfy all requests coming from different clients and manage the computing resources. For efficient resource management, the cloud service provider uses different types of scheduling techniques [4]. All the operations on cloud computing are enormously affected by the allocation and scheduling of resources. Therefore, many researchers have shown interest in task scheduling and resource allocation in the cloud [5,6].

A task needs to be completed before the deadline to achieve the quality of service (QoS) and satisfy the service level agreement (SLA) [7,8]. Load balancing plays an essential role in achieving better performance in cloud systems [9,10]. The users' tasks must be distributed among all available computing nodes so that no computing node will be overloaded or under-loaded [11,12]. If a node is overloaded, users' tasks may unnecessarily miss their deadline in the waiting queue. Similarly, if a node is under-loaded, there will be a wastage of computing resources putting the computing node in an idle state. Hence, proper use of the load-balancing algorithm will reduce response time and improve user-satisfaction [13,14].

Different approaches are there for load balancing in cloud computing, such as using an efficient task scheduling algorithm, efficient resource allocation, resource migration such as VM or container, resource reservation [15], and service migration [16], etc.

In addition to a typical cloud service model such as IaaS, PaaS, and SaaS, a new cloud service model Containers as a Service (CaaS) has really been introduced by different cloud service providers. A Container in cloud computing is an approach to operating system-level virtualization [17]. The application code, runtime, system tools, system libraries, and settings are all included in a container image, which is a lightweight, independent, executable bundle of software. It contains everything you need to execute the program. Containers are the fundamental building blocks of operating system-level virtualization [18]. They provide isolated virtual environments without the need of an intermediary monitoring medium such as a hypervisor. OS level virtualization can be implemented using OS Container or Application Container [19]. In this paper, we consider the OS container for our experiment. OS containers are best suited when you want to package different libraries, languages, databases, etc. Application container is best suited when you want to package the application as a component [20]. The architecture of a virtual machine and a container is illustrated in Figure 1.

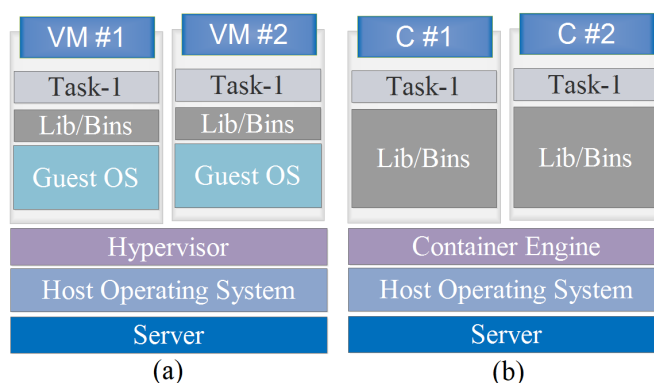


Figure 1. Virtualization Technology. (a) VM Architecture. (b) Container Architecture.

The Container as a Service (CaaS) model implements hybrid virtualization. In CaaS, tasks get executed in a container scheduled in VMs, and VMs are hosted in a physical machine. The architecture of a CaaS model shown in Figure 2 is a combination of system and OS-level virtualization. In a CaaS model, the system allows users to run, manage, scale, upload, and organize containers using OS-level virtualization. It is highly portable, i.e., once you create a container for an application or task, this container has everything to run that application. One of the main reason of using the CaaS model is security. Virtual machines are more isolated than containers. So, instead of running all containers in a server, if we group them and deploy in different virtual machines, they will be more isolated and secure. It allows the user to run that application in a different private or public cloud environment. This allows the user to switch from one cloud provider to another without any difficulty. CaaS is positioned in the middle between Infrastructure as a Service (IaaS) and platform as a service (PaaS) in the hierarchy of cloud computing services. CaaS is considered a subset of Infrastructure as a Service (IaaS). Two popular CaaS orchestration platforms are Google Kubernetes and Docker Swarm.

The main objective of task scheduling is to efficiently utilize all the resources and organize all the incoming requests so that all requests complete their execution before the deadline. Multiple tasks may be executed simultaneously in a cloud computing system, and the same resource is allocated to several clients. So, if the system does not use a proper scheduling algorithm, smaller tasks may miss their deadline in the waiting queue for a long time. The main contribution of this paper is as follows:

- Proposed a Grey Wolf Optimization (GWO) based Simulated Annealing approach to counter the problem of load balancing in the containerized cloud that also considers the deadline miss rate.
- The Simulated Annealing algorithm was implemented along with Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Grey Wolf Optimization (GWO).
- The efficiency of each algorithm has been compared in terms of load variation and makespan.

The rest of the paper is organized as follows. Section 2 describes the related work on load balancing and resource allocation in the cloud. In Section 3, we represent different components of CaaS cloud architecture such as container mapper, container scheduler, resource manager, migration manager, etc. We present our proposed system model and algorithms in Section 4. Section 5 represents the experimental analysis and results. Finally, in Section 6, we draw the conclusion with some future directions.

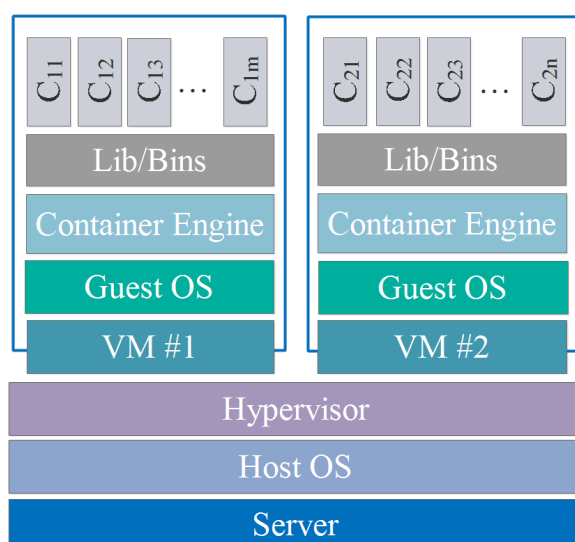


Figure 2. Container as a Service Model.

2. Related Work

There are many different types of meta-heuristics that may be used in the search for the best answer to a wide variety of optimization issues [21]. Many meta-heuristic algorithms have recently been effectively used to solve optimization issues. Solving complicated problems with meta-heuristic techniques is appealing because they provide good solutions even for extremely large problems in short periods of time. In this section, we review and represent the related works on load balancing in cloud computing, which are similar and relevant to our work. Related research is grouped around meta-heuristic algorithms inspired by load-balancing approaches that have been developed for cloud computing environments, as well as hybrid optimization algorithms that are based on variations of GA, PSO, and GWO.

Effective task scheduling and load distribution across all processing nodes may reduce response, execution, and waiting time, boosting cloud system performance. Several GA-based load-balancing strategies listed in Table 1 have been presented. The I-GA, an improved genetic algorithm is presented to solve the virtual machine placement issue in a cloud data center to increase availability and energy usage [22]. In [23], the author proposed an algorithm for load-balanced task scheduling, job spanning time, and load-balancing genetic algorithm (JLGA) that considers a double-fitness adaptive algorithm. In [24], an optimized protocol based on genetic algorithm is designed for cluster head selection. Due to the high number of retransmissions induced by packet loss, the data transmission latency of some smart meters is considerable. A genetic algorithm is presented to avoid this issue and improve the end-to-end latency in [25]. For scheduling tasks in a cloud environment,

a Multi-Population Genetic Algorithm (MPGA) that takes load balancing into account is used in [26]. In [27], for multimedia applications, the author offered a novel genetic load-balancing technique that distributes the load across servers and thereby minimizes the response time to the users. Authors in [28] proposed GA-OA (genetic algorithm-based adaptive offloading) for efficient traffic management in an IoT-infrastructure-cloud scenario. Several other GA-based approaches have been proposed for load balancing in the cloud in [29–31].

The Particle Swarm Optimization is another type of nature-inspired algorithm that has been proposed for resource scheduling and load balancing in cloud systems by many researchers [32–34]. To allocate resources for the execution of different tasks, Arabinda Pradhan and Sukant Kishoro Bisoy in [32] proposed a modified PSO algorithm that minimizes the makespan, maximizes resource utilization, and at the same time performs load balancing. In [33], Ronak Agarwal et al. proposed a mutation-based PSO algorithm for load balancing. The main idea of this algorithm is to minimize the makespan and improve the fitness function. The experimental result indicates that the proposed algorithm performs better than a normal PSO algorithm. In [34], authors proposed the FIMPSO algorithm for load balancing and energy efficiency. To minimize the search space, a Firefly FF algorithm is proposed and for an enhanced response, the IMPSO algorithm has been proposed. The parameter considered for evaluation were throughput, makespan, execution time, and resource utilization. Authors in [35,36] introduced a strategy that ensures that tasks that cause VM overload are transferred to comparable VMs in the cloud system in the most efficient manner possible. The proposed optimization model's aim functions are task execution and transfer time minimization. Minimizing transfer time and execution time is their main objective. For independent and non-preemptive tasks in the cloud, a PSO-based static task scheduling approach has been proposed in [37]. In [38], the authors emphasized how considering only execution time can lead to workload imbalance in a virtual machine. To solve this problem, a new task scheduling approach based on PSO has been proposed. How a PSO-based algorithm suffers due to random movement of particles is highlighted in [39]. They designed Pbest discrete PSO to solve these issues. Several approaches using PSO are presented in Table 2.

Another meta-heuristic approach for distributing workload among all servers equally is Grey wolf optimization (GWO). Proposer distribution of work load improves the productivity of cloud systems and users' satisfaction. Authors in [40] proposed a GWO-based algorithm to maintain load balancing considering resource reliability capacity. The suggested algorithm looks for nodes that are currently not in use and then attempts to determine each node's thresholds and fitness function to assign the load. For a containerized cloud system, a GWO-based load-balancing algorithm has been proposed in [41]. The simulation results indicates that the GWO-based algorithm outperforms GA and PSO. One of the essential tasks to maximize the VM utilization in the cloud is optimal workflow scheduling. All dependent tasks of a workflow need to be distributed among the available VMs in order to complete their execution. A discrete variation of distributed GWO has been introduced in [42] for workflow scheduling. Many event-driven integrations and cloud services are available with serverless computing, making them simple to create and execute while also allowing for sophisticated cost management. The serverless runtimes can only be used for low data and storage applications, such as machine learning prediction and inference. However, these applications have been enhanced over other cloud runtimes. In [43], GWO is applied to enhance the workload distribution and to parallelize the serverless event queue and dispatcher, an ML model is proposed. To address the energy hole problem, Grey Wolf Optimization (GWO) is used for WSN clustering and routing to save energy. In addition, two new fitness functions for grouping and routing issues are provided. To reduce the total distance traveled and the number of hops, a fitness function is designed in [44]. Several approaches using GWO are presented in Table 3.

Table 1. Review of different Meta-heuristic Load-Balancing Approaches Using GA.

Author(s)	Method	Objective of Work	Remark
[22]	GA	Efficient virtual machine placement in cloud data center	Considered only memory and CPU. Other aspect of energy consumption with large problem needs to be considered.
[23]	GA	Scheduling of tasks to ensure the shortest possible makespan and load balancing.	Other static parameters in GA, such as population scale, iterations, crossover, and mutation operator, could potentially exert a negative influence on the rate of convergence and the quality of the global optimal solution.
[24]	GA	The Genetic Algorithm-based Optimized Clustering (GAOC) technique was developed specifically for the purpose of optimizing CH selection.	Work could be extended for the shifting sink situation in order to achieve higher levels of performance.
[25]	GA	GA-based optimization approach was used for load-balanced path	Other constraints such as energy, resource usage can be considered for further improvement.
[26]	GA	The Multi-Population Genetic Algorithm, or MPGA, is a scheduling method for tasks that takes load balance into consideration.	Coexistence of independent and related tasks in practical cloud environment.
[27]	GA	Technique for load balancing used in multimedia applications, with the goal of increasing performance by dispersing the workload among multiple servers	In the future, one of the most important tasks for our study area will be to demonstrate the usefulness and practicality of SDN in blockchain.
[28]	GA	GA-OA is an acronym that stands for genetic algorithm-based adaptive offloading. It is used for optimal traffic handling in environments involving IoT infrastructure and the cloud.	It is possible to use an adaptive offloading approach that has been strengthened with intelligent learning algorithms in order to extend the flexibility and scalability features present in a heterogeneous network.

Table 2. Review of different Meta-heuristic Load Balancing Approaches Using PSO.

Author(s)	Method	Objective of Work	Remark
[32]	PSO	A modified PSO algorithm that minimizes the makespan, maximizes resource utilization, and performs load balancing	Work can be extended to improve the quality of service
[33]	PSO	Mutation-based PSO for load balancing in a cloud data center	Work can be extended by considering parameters such as waiting time, throughput, etc.
[34]	PSO	FIMPSO algorithm for load balancing and energy efficiency	Proposed FIMPSO method can be improved to make use of data deduplication
[36]	PSO	Tasks are migrated from overloaded VM to under-loaded VM. The proposed PSO-based algorithm minimizes task execution and transfer time	Work can be extended by considering execution time, makespan, etc.
[37]	PSO	PSO-based static task scheduling algorithm in which it is assumed that the tasks are non-preemptive and independent of each other.	Work can be increased to accommodate workflow applications while also taking into account other quality of service parameters such as fault tolerance and cost minimization.

Table 3. Review of different Meta-heuristic Load Balancing Approaches Using GWO.

Author(s)	Method	Objective of Work	Remark
[42]	GWO	A discrete variation of distributed GWO has been introduced for workflow scheduling	Work could be expanded to examine the performance of DGWO employing intricate scientific workflow apps hosted in the cloud.
[43]	GWO	A machine learning approach for distributing tasks in a serverless framework to the event queue and the dispatcher. To enhance workload distribution, GWO is applied.	In the not too distant future, task allocation to distributed deep learning may be developed as a means of achieving improved network traffic analysis.
[44]	GWO	To address the energy hole problem, Grey Wolf Optimization (GWO) is used for WSN clustering and routing to save energy.	Sensor nodes, gateways, and base stations may be deliberately relocated in order to study the impact of location shift on these devices in the future.
[41]	GWO	Proposed a GWO-based algorithm for containerized cloud	Work can be expanded by considering other parameter such as waiting time, response time, etc.
[40]	GWO	GWO is used to maintain load balancing by considering resource reliability capability.	Work can be expanded by involving other factors such as accessibility, security, and scalability, etc. In addition to this, dependent tasks can be considered for dynamic load balancing to reliability.

3. Architecture for CaaS Cloud

An overview of the CaaS cloud architecture is shown in Figure 3. It is divided into two main parts: the user side and the cloud system. The user side consists of different types of jobs: user application, user task, data analytics, transaction processing, image processing, etc. Each job in the CaaS cloud architecture comprises several independent tasks that are executed in a container. For each independent task, a separate container is allocated. On the other hand, the cloud system facilitates all hardware and software requirements and provides computing capability, storage, and networking for data sharing. There will be more than one server available in the cloud system, which is responsible for providing resources to a virtual machine. All virtual machines are hosted in servers, and on top of it, all containers are deployed in virtual machines. The number of VMs a server can accommodate depends on the server’s available memory size and processor core. Similarly, the number of containers a VM can run depends upon the available resource of the VM [45]. The interaction among the main components of the CaaS cloud architecture is described in Figure 4. In the CaaS cloud architecture, the cloud system has four main components.

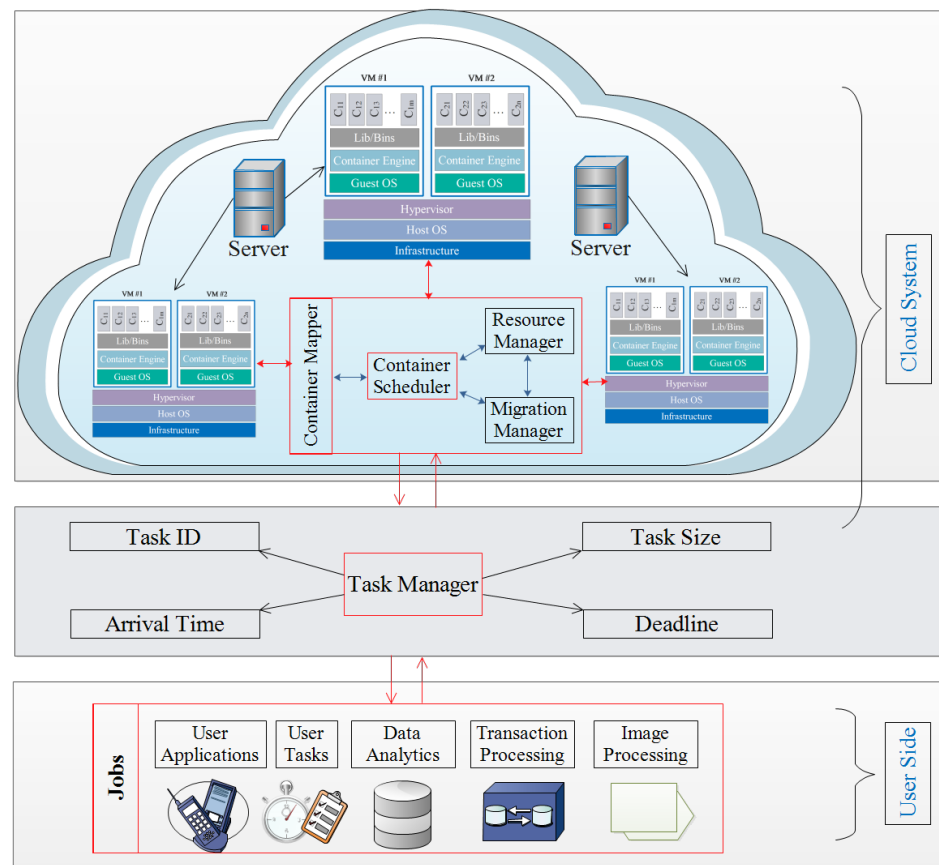


Figure 3. Architecture for CaaS Cloud.

Container Mapper: The container mapper accepts users’ tasks from the task manager and prepares a container by packaging all the binaries and libraries required for it. The task gets executed in that container. The main responsibility of the container mapper is placing a container into a suitable VM or PM. An example container placement mechanism is depicted in Figure 5. For example, three servers or physical machines are there in the cloud. Server-1 has instantiated two VMs, VM-1 and VM-2. In VM-1, two containers C1 and C2 are running. In VM-2, there is only one container, C3, running. Similarly, in Server-2, two VMs are instantiated, each running one container. In Server-3, two VMs are instantiated, but only one VM is running one container. Now, suppose C7 is the new container that needs to be

placed in any VM. The container scheduler will take this decision and determine a suitable VM for it, and the container mapper will place that container in the selected VM.

Resource Manager: The resource manager maintains and keeps track of the information about both physical machines and virtual machines, as shown in Figure 6. The Physical Machine Controller (PMC) available in each physical machine maintains information about how many VMs and types of VMs are instantiated in a physical machine and available resource information. Similarly, the virtual machine controller (VMC) is a demon present in each virtual machine responsible for maintaining resource information in a VM, such as the number of containers running and available resources.

Migration Manager: The migration manager is responsible for migrating VMs from one PM to another. Migrating a VM is to improve resource utilization by evenly distributing instantiated VMs among all available PMs and minimizing the active PMs. VM migration takes place from overloaded PMs to lightly loaded PMs. If a PM X is active with a minimal workload, we can migrate the workload of X to another moderately loaded PM Y and turn off the PM X to minimize the number of PMs. The migration manager immediately updates the migration information to the resource manager when there is any VM migration. Similarly, a container can be migrated from one VM to another to balance the workload among VMs. A container can be migrated from one VM on server X to another VM on server Y, termed interPM container migration. If a container is migrated from one VM to another VM in the same physical machine, it is called intraVM container migration. PMC and VMC are responsible for updating VM and container migration information, respectively, to the migration manager.

Container Scheduler: The container scheduler determines which container will be deployed in which virtual machine. This plays an important role in our objective of distributing the user tasks evenly among all servers using the GWO-SA algorithm. Each job in the CaaS cloud architecture consists of several independent tasks. Each task gets executed in a separate container. Whenever a task arrives for execution, the container scheduler communicates with the resource manager to obtain the currently available resource information of each PM and VMs. When the container scheduler finds a VM with sufficient resources for container placement, it conveys the information to the container mapper. Then, the container mapper contacts the VMC of that VM to launch the container. If a suitable VM is not found, the container scheduler will contact PMC to instantiate a new VM and pass that VM’s information to the container mapper. Then, the container mapper contacts the VMC of the newly created VM to launch the container.

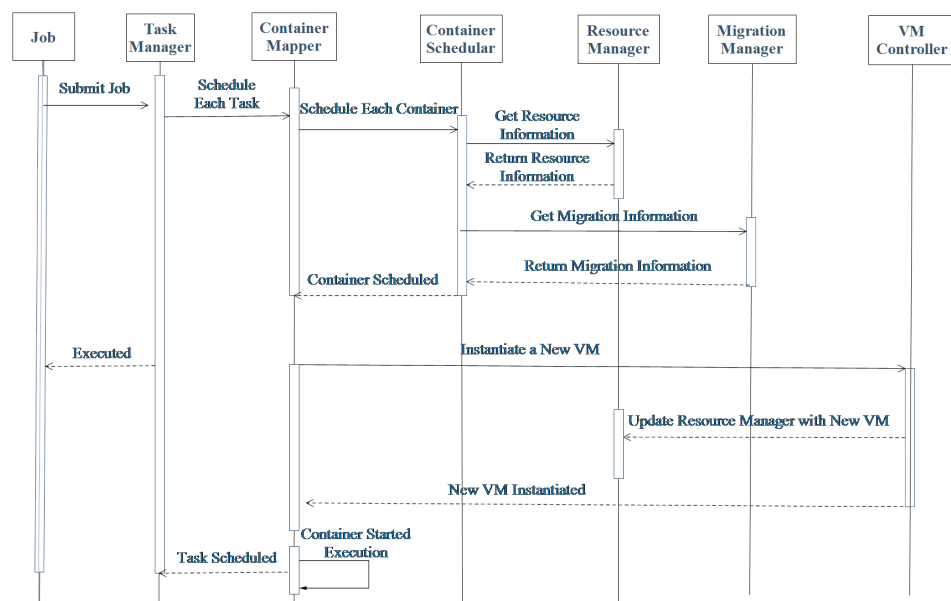


Figure 4. Sequence Diagram for Scheduling a Container in the CaaS Cloud.

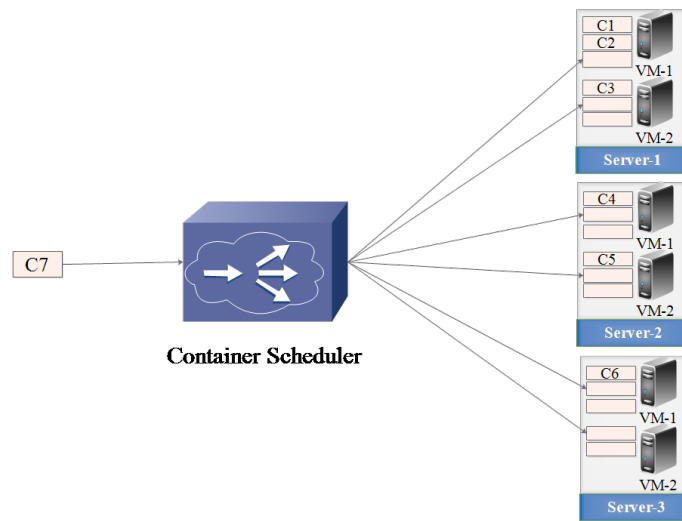


Figure 5. Example of Container Placement.

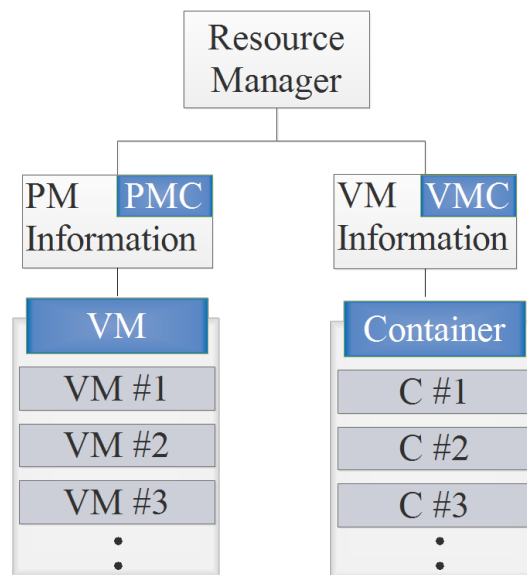


Figure 6. Resource Manager.

4. System Model

This section presents a detailed description of the proposed system model and the terminology. It consists of five main components: Server Model, VM model, Container Model, Task Model, and Scheduling Model. The description of symbols used are presented in Table 4.

4.1. Server Model

Servers are the physical machines responsible for providing hardware and software resources over the internet. Let $S = \{s_1, s_2, \dots, s_m\}$ be a set of servers where each server s_j , $j = 1, 2, 3, \dots, m$ is an independent server with certain memory and CPU core.

4.2. VM Model

A virtual machine is an instance of a server responsible for execution of a container on it. Let $V = \{v_1, v_2, \dots, v_l\}$ be a set of virtual machines where each virtual machine v_l , $l = 1, 2, 3, \dots, p$ is an independent VM with certain memory and CPU core to run containers on it.

Table 4. Description of Symbols Used.

Symbols	Description
S	Set of servers
s_j	j th server
V	Set of virtual machine
v_l	l th virtual machine
C	Container
C_{kj}	k th container on j th server
$MemS_{kj}$	Memory size of k th container on j th server
$ProcS_{kj}$	Processing speed of k th container on j th server
T	Set of tasks
t_i	i th task
T_{id}	Unique task ID
AT_i	Arrival time of i th task
TS_i	Size of i th task in terms of Million Instruction(MI)
DL_i	Deadline of i th task
ET_{ikj}	Expected execution time of i th task on k th container of j th server
CT_i	Expected completion time of i th task
CT_{ikj}	Expected completion time of i th task on k th container of j th server
WT_i	Total waiting time of the i th task
MS	Makespan
L_{var}	Variance of load factor of all the servers
L_j	Load factor of j th server
μ	Mean of load factors of all the servers

4.3. Container Model

The containerized cloud system is characterized by a set of servers $S = \{s_1, s_2, \dots, s_m\}$ that provides the infrastructure for creating containers by OS virtualization. The containers pull the required libs and bins from the host OS running in the server to run an application. Each server is having a certain amount of RAM in GB and a dedicated processing capability [46] in million instructions per second (MIPS). So, each container is modeled by three parameters $C = \{C_{kj}, MemS_{kj}, ProcS_{kj}\}$, where C_{kj} represents the k -th container on the j -th server, $MemS_{kj}$ is the memory size of the k -th container on the j -th server, and $ProcS_{kj}$ is the processing speed of the k -th container on the j -th server which are totally dependent on the configuration of the server. One of the most popular container management systems is Docker, which adopts a client–server architecture [47]. A Docker demon manages all the containers in a server, and whenever a request arrives at the Docker demon, it will allocate the necessary container. More than one Docker demon can be run simultaneously on a single server resulting in massive horizontal scalability. However, in general, we focus on multiple processor cores on a single server and only one Docker demon running and managing all containers [48].

4.4. Task Model

There are n number of tasks arriving dynamically from different users and the set of the task is represented by $T = \{t_1, t_2, \dots, t_n\}$. Each task t_i in T is having four attributes; $t_i = \{T_{id}, AT_i, TS_i, DL_i\}$, where T_{id} is the unique task ID, AT_i represents the arrival time of the i -th task, and TS_i is the task size (in terms of Million Instruction(MI)) and DL_i is the deadline of the i -th task [46]. Since the tasks are heterogeneous, the expected execution time ET_{ikj} of the i -th task on the k -th container of the j -th server can be calculated as

$$ET_{ikj} = \frac{TS_i}{ProcS_{kj}} \quad (1)$$

The expected completion time CT_{ikj} of the i -th task on the k -th container of the j -th server can be calculated as

$$CT_{ikj} = WT_i + ET_{ikj} \quad (2)$$

where WT_i is the total waiting time of the i -th task.

The total time taken by all the tasks to complete their execution is called makespan (MS) and is calculated as

$$MS = \sum_{i=1}^n CT_i \quad (3)$$

The objective is to minimize MS as much as possible.

The load distribution among the servers is evaluated by calculating the variance of the load factor of each server.

$$L_{\text{var}} = \frac{\sum_{i=1}^m (L_j - \mu)^2}{m} \quad (4)$$

where, L_{var} is the variance of load factor of all the servers, L_j is the load factor of the j -th server and μ is the mean of load factors of all the servers and m is the total number of servers.

4.5. Scheduling Model

The primary purpose of the scheduling model is to receive the tasks and map them to an appropriate server for the execution by considering the load factor and expected completion time while minimizing the makespan. The scheduling algorithm gets executed after a specific time interval and maps all the tasks that arrived. Some of the salient features of the scheduling model are:

- It considers the number of VMs and containers running in a server and compares it with the maximum number of containers possible in that server.
- It considers the load factor of all the servers and selects an appropriate server in which the container will be deployed in a VM.
- It also takes care of the maximum time taken by individual servers to complete a task and tries to ensure that the task will be completed before its deadline.

Task generators are responsible for generating different types of tasks. Since the tasks are of different types, their resource requirement is also different. Once the task is generated, they are forwarded to the task manager, responsible for screening and maintenance of all the tasks. The task manager maintains a proper order among tasks and sends them for scheduling. The task scheduler is responsible for selecting an appropriate VM in a server to execute the task. The container mapper creates a container image on a VM on which the task will be executed and started. The number of containers that can be deployed in a VM depends on the number of processor cores.

4.6. Dataset

We have made use of the Google cluster trace dataset that was made public in May 2019 by Google. Google clusters consist of many computers linked by a specialized, high-bandwidth network and housed in racks. The term "cell" refers to a group of computers that are all part of the same cluster and use the same cluster management system to divide up the workload. This is a trace of the workloads which was operating on eight different compute clusters belonging to Google Borg. The trace provides an account of each task submission, scheduling decision, and details regarding resource use for each job that was executed in those clusters.

A job in Borg is a collection of tasks that define the computations a user wishes to do. In contrast, an alloc set is a collection of allocs or alloc instances that indicate a resource reservation in which jobs may be executed. Linux applications are represented by tasks, each of which may consist of many processes. All of a job's tasks will execute in (get resources from) an alloc instance belonging to the alloc set the job has specified. Without an alloc set, a job's tasks will use the machine's resources without allocating them. Things

may be either a collection (such as a group of jobs or allocs) or an instance (such as a single task or alloc).

A single-use trace may represent the amount of work done on a single Borg cell over the course of many days. A trace is constructed up of a number of tables, each of which is indexed by a primary key, which will normally consist of a timestamp. The information that is included in the tables was obtained from the management system of the cell as well as the various machines that were located inside the cell.

Data Tables

In this section, some of the important tables that are included inside the traces will be discussed.

- **Machines**
The MachineEvents table and the MachineAttributes table are used to describe machines. These tables may be found in the machine description. The machine event table may include one or more entries for each machine to provide a description of that machine. The vast majority of data discusses computers that were already operational when the trace was initiated. The kernel version, clock speed, existence of an external IP address, and other machine features are examples of machine attributes. Machine attributes are key-value pairs that reflect the qualities of the machine. It is possible for instances to place limitations on machine characteristics.
- **Machine Event Table**
The table labeled “Machine Event” provides an explanation of each of the machines. It includes details such as the timestamp when the machine was launched, the identification number of the machine, and the different types of events, which include ADD, REMOVE, and UPDATE, Platform ID, the capacity of the CPU, and the amount of memory.
- **Machine Attribute Table**
Machine attribute values are supplied as integers or encrypted (hashed) texts depending on the number of possible values for a given attribute name. In a first scenario, the values of characteristics are recorded and sorted (in numerical order, if all are numbers) after being seen across all machines. One corresponds to the first value, two to the second, and so on. Attributes whose existence or absence alone serves as a signal (such as whether or not a computer is executing a certain software) are assigned the value one if they are present.
- **Collection Event Table**
Collection-related events are summarized in this table. The following descriptions apply to each field. The first group is shared by collections and instances but with a significantly context-dependent meaning. At any given time in the trace, the thing event tables will include information about all things that are either now being executed (marked as RUNNING) or are schedule-eligible but have not yet been scheduled (marked as PENDING). We will include a record for each collection instance in the trace, which will provide the collection’s scheduling restrictions. Scheduling_class and priority are the two important attributes in this table. Tasks and jobs are classified according to their sensitivity to delays in execution by a scheduling class. One number represents the scheduling class, with 3 being a more time-sensitive job and 0 indicating a non-production job. Priorities are assigned to everything, and those integers are translated into a scale from 0 (the lowest) to 360 or more (the highest). Higher-priority items often get more preference than lower-priority items.
- **Instance Event Table**
The data in this table pertain to occurrences that have occurred in instances (tasks and alloc instances). Time, scheduling class, priority, and so on, are only a few of the initial sets of fields that are also included in the CollectionEvents table. Instance-specific attributes may consist of machine id, resource request, and others. Resource requests show how much memory and CPU an instance can use (called its limit). Tasks that are used too much may be slowed down or stopped (for resources such as memory).

Because the scheduler may over-commit system resources, there may not be enough to meet the needs of all tasks during their run-time, even if each task is not used as much as it could be. This could kill a low-priority job(s).

- **Instance Usage Table**

The amount of resource utilization that is reported for an alloc instance is the amount of resource utilization for all of the jobs that are executed inside it during each sample period. This table contains 18 fields including `cpu_usage_distribution`, `tail_cpu_usage_distribution`, and `cycles_per_instruction`. The `cpu_usage_distribution` and `tail_cpu_usage_distribution` vectors each give in-depth information on the distribution of the amount of NCUs used by the CPU throughout the course of the five-minute measurement frame. The statistics known as Cycles Per Instruction (CPI) and Memory Accesses Per Instruction (MAI) are gathered from the performance counters of the CPU; however, not all machines get this information. Memory accesses are determined by the measurements of the cache level most recently used.

4.7. Proposed Algorithm

Here, we have presented our proposed meta-heuristic based algorithms for distribution of load among all available servers evenly and minimizing the number of servers. Algorithm 1 is used to calculate the fitness value and is called by each algorithm. Algorithm 2 represents the proposed GWO-based algorithm, and Algorithm 3 represents the simulated annealing approach. This is used with all three algorithms: Genetic Algorithm [49], Particle Swarm Optimization [50], and proposed Grey Wolf Optimization.

Algorithm 1 Fitness Function

Input: `solution_list`, `population_size`,

Output: fitness value.

- 1: Initialize $LV = 0$, $MS = 0$.
 - 2: **for** each solution in `solution_list` **do**
 - 3: Calculate $MS = \sum_{i=1}^n CT_i$.
 - 4: Calculate $L_{var} = \frac{1}{m} \sum_{i=1}^m L_j - \mu$.
 - 5: **end for**
 - 6: return MS , L_{var} .
-

Grey Wolf Optimization Algorithm

Grey Wolf Optimization is a meta-heuristic algorithm based on population and it imitates the hierarchy of a leadership and hunting mechanism of gray wolves in nature. They prefer to live in a pack that is divided into four different levels. The name of the level-1, level-2, level-3, and level-4 are alpha, beta, delta, and omega, respectively. Level-1 is the leader responsible for making a decision and it can be male or female. The level-2 category is the disciplinarian and adviser for the pack and assists the level-1 group in decision-making. The level-3 is consisting of scouts, sentinels, elders, hunters, and caretakers. The scouts are responsible for watching the boundaries, the pack is supposed to be protected by sentinels, elders are the ones who were sometimes alpha or beta, hunters are the ones who assist alpha in hunting, caretakers are responsible for taking care of weak, ill, and wounded wolves. Level-4 groups are scapegoats in the pack which are the last-allowed wolves for eating. According to the position of alpha, beta, and delta wolves, all other wolves search for prey. The best solution is produced by alpha, the second-best solution is the beta solution, and then the delta solution. All other remaining solutions produced by omega are categorized as omega solutions.

To update the position, the following mathematical formulas are used by the omega group:

$$\vec{D} = |\vec{Q} \cdot S_p \vec{i} - \vec{S}(i)| \quad (5)$$

$$\vec{S}(i+1) = S_p \vec{i} - \vec{P} \cdot \vec{D} \quad (6)$$

where \vec{P} and \vec{Q} are coefficient vectors and i represents the current iteration number, \vec{S} represents the position vector of grey wolf and \vec{S}_p indicates the position vector of the prey. The vectors \vec{P} and \vec{Q} can be calculated as

$$\vec{P} = 2 \cdot \vec{a} \cdot \vec{r}_1 - \vec{a} \tag{7}$$

$$\vec{Q} = 2 \cdot \vec{r}_2 \tag{8}$$

where \vec{r}_1 and \vec{r}_2 are random vectors. The vector component \vec{a} is linearly decreased from 2 to 0 using

$$a = 2 - \frac{2 \cdot i}{\text{Number of Iteration}} \tag{9}$$

where i is the current iteration number.

The obtained solution by the Grey Wolf optimization Algorithm is represented as $S = \{2, 4, 3, 7, 5\}$, where each entry in S represents the container number. If the obtained solution is $S = \{2, 4, 3, 7\}$, it says that task t_1 is assigned to container C_2 , task t_2 is assigned to container C_4 , task t_3 is assigned to container C_3 , and task t_4 is assigned to container C_7 . Then, based on the available VM and this task allocation to different containers, makespan and load variation are calculated. The three best solutions which minimize the load variation and makespan are considered as alpha, beta, and delta solutions. Based on the position of alpha, beta, and delta, the remaining solutions are updated. The flow of the algorithm is presented in Figure 7.

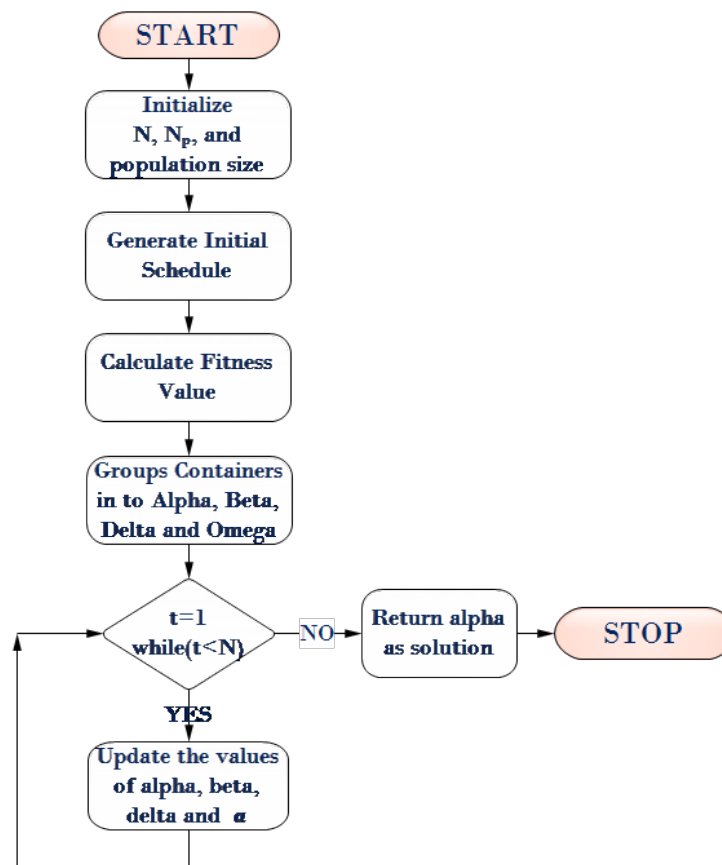


Figure 7. Flowchart of the Proposed Algorithm.

Algorithm 2 Grey Wolf optimization Algorithm

Input: N, N_p, p_size
Output: *solution*.
 $N \leftarrow$ Number of Iteration.
 $N_p \leftarrow$ Number of population.
 $p_size \leftarrow$ Population Size.
 $N_t \leftarrow$ Number of task.
 $N_c \leftarrow$ Number of Container.
 $fit_val \leftarrow$ Fitness value vector.
 $alpha \leftarrow$ alpha wolf
 $beta \leftarrow$ beta wolf.
 $delta \leftarrow$ delta wolf.
 $alpha_fit_val \leftarrow$ Fitness value of alpha wolf.
 $beta_fit_val \leftarrow$ Fitness value of beta wolf.
 $delta_fit_val \leftarrow$ Fitness value of delta wolf.
 $pop \leftarrow$ population matrix of size(N_p, p_size).
 $P \leftarrow$ Coefficient Vector.
 $Q \leftarrow$ Coefficient Vector.
 $r_1 \leftarrow$ Random Constant.
 $r_2 \leftarrow$ Random Constant.
 $solution \leftarrow$ represents solution of the problem.

- 1: $pop \leftarrow \phi$.
- 2: **for** $i \leftarrow 1$ to N_p **do**
- 3: $temp \leftarrow create_chromosome(N_c, N_t)$
- 4: $pop \leftarrow pop \cup temp$.
- 5: **end for**
- 6: **return** *solution*.
- 7: **for** $t \leftarrow 1$ to N **do**
- 8: $fit_val \leftarrow Fitness(pop)$.
- 9: Sort the pop according to fit_val in ascending order.
- 10: $alpha \leftarrow pop[0]$.
- 11: $alpha_fit_val \leftarrow fit_val[0]$.
- 12: $beta \leftarrow pop[1]$.
- 13: $beta_fit_val \leftarrow fit_val[1]$
- 14: $delta \leftarrow pop[2]$.
- 15: $delta_fit_val \leftarrow fit_val[2]$.
- 16: **for** $i \leftarrow 2$ to N_p **do**
- 17: $a \leftarrow 2 - i \cdot (2/N_p)$.
- 18: $r_1 \leftarrow random()$
- 19: $r_2 \leftarrow random()$
- 20: $P \leftarrow 2 \cdot a \cdot r_1 - a$
- 21: $Q \leftarrow 2 \cdot r_2$
- 22: $D \leftarrow | Q \cdot X - pop[i - 1] |$ ▷ X is $alpha, beta, delta$
- 23: $pop[i] \leftarrow X - P \cdot D$ ▷ X is $alpha, beta, delta$
- 24: **end for**
- 25: **end for**
- 26: Sort the pop according to fot_val in ascending order.
- 27: $solution \leftarrow pop[0]$.
- 28: **return** *solution*.

Algorithm 3 Simulated Annealing Algorithm

Input: $N, T_0, T_{end}, alpha$
Output: *solution*.
 $N \leftarrow$ Number of Iteration.
 $N_t \leftarrow$ Number of task.
 $N_c \leftarrow$ Number of Container.
 $fit_val \leftarrow$ Fitness value.
 $T_0 \leftarrow$ Initial Temperature.
 $T_{end} \leftarrow$ Ending Temperature.
 $new_fit_val \leftarrow$ New fitness value.
solution \leftarrow represents solution of the problem.

- 1: Initialize *solution* randomly.
- 2: $fit_val \leftarrow$ Fitness(*solution*).
- 3: **while** $T_0 > T_{end}$ **do**
- 4: **for** $i \leftarrow 1$ to N **do**
- 5: $rnd_pos \leftarrow$ random(0, N_t).
- 6: $rnd_num \leftarrow$ random(0, N_c).
- 7: $temp \leftarrow$ *solution*.
- 8: $temp[rnd_pos] = rnd_num$.
- 9: $new_fit_val \leftarrow$ Fitness(*solution*).
- 10: **if** $new_fit_val < fit_val$ **then**
- 11: $fit_val \leftarrow new_fit_val$.
- 12: $solution \leftarrow temp$.
- 13: **else**
- 14: $delta \leftarrow e^{-\frac{new_fit_val - fit_val}{T_0}}$
- 15: $rnd \leftarrow$ random().
- 16: **if** $delta > rnd$ **then**
- 17: $fit_val \leftarrow new_fit_val$.
- 18: $solution \leftarrow temp$.
- 19: **end if**
- 20: **end if**
- 21: **end for**
- 22: $T_0 \leftarrow T_0 \cdot alpha$.
- 23: **end while**
- 24: **return** *solution*.

5. Experimental Analysis and Results

The list of parameters considered to perform our experiment with the proposed system model is presented in Table 5. All the experiments were conducted in a non-elastic environment. To fix the number of iterations for optimization algorithm, we perform our experiment with a varying number of containers from 10 to 50 and number of tasks from 1000 to 3000 and considering 1000 iterations. A constant value was observed after 1000 iterations for load variation and makespan. Therefore, for our proposed algorithm, we fix the number of iterations to 1000.

The initial population size has been considered to be 1000. We performed our experiment by varying different parameters and observed the results. In most cases, more than 97% of the tasks were meeting their deadline, i.e., they could finish their execution before the deadline. We have considered two important parameters for our experiment: load variation and makespan. The makespan is the total time taken to complete the execution of all the tasks. The variance is calculated to observe the load variation. If the variance is low, it indicates the even distribution of the workload. The results of the six different approaches have been compared using these two parameters, and we found that the Grey wolf optimization algorithm with simulated annealing is performing better than all other approaches.

The results of all the different approaches are shown in Tables 6 and 7. The results are shown graphically in Figures 8–13. Figure 8 depicts the variation of load with 1000 tasks by considering 10, 20, 30, 40, 50 containers. Figure 10 depicts the variation of load with 2000 tasks by considering 10, 20, 30, 40, 50 containers. Figure 12 depicts the variation of load with 3000 tasks by considering 10, 20, 30, 40, 50 containers. Figure 9 depicts the makespan of different approaches with 1000 tasks by considering 10, 20, 30, 40, 50 containers. Figure 11 depicts the makespan of different approaches with 2000 tasks by considering 10, 20, 30, 40, 50 containers. Figure 13 depicts the makespan of different approaches with 3000 tasks by considering 10, 20, 30, 40, 50 containers.

The variations of results obtained using different approaches for load variation are presented in Figure 14, and makespan is presented in Figure 15.

Table 5. Parameters Considered for the Proposed System Model.

Parameters	Unit
Number of Containers	10, 20, 30, 40 and 50
Speed of Container	2500 to 4000 MIPS
Arrival Time of Task	0 to 1000 unit
Size of Task	5000 to 10,000 MI
Deadline of Task	AT + U (5, 10)
Number of Task	1000, 2000, 3000

Table 6. Load Variations.

Algorithms	1000 Tasks	2000 Tasks	3000 Tasks
GA	3.22631	3.63370	4.94797
GA-SA	3.23531	3.53470	4.84397
PSO	3.29918	3.79661	4.74392
PSO-SA	3.12431	3.41361	4.64282
GWO	3.89636	3.81661	4.79862
GWO-SA	2.72631	2.41361	4.45271

Table 7. Makespan.

Algorithms	1000 Tasks	2000 Tasks	3000 Tasks
GA	2180.68	2472.19	3628.95
GA-SA	1978.58	2162.53	3438.95
PSO	2266.57	2882.11	3717.87
PSO-SA	1962.84	2647.38	3372.95
GWO	2298.96	2798.94	3872.82
GWO-SA	1951.34	2138.13	3217.95

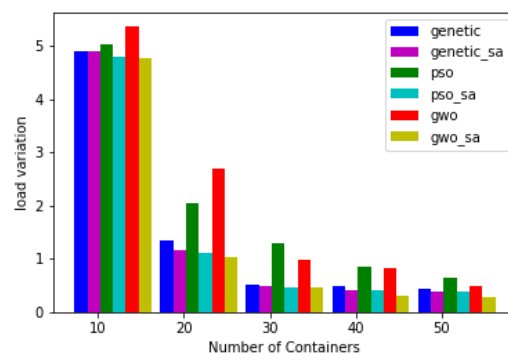


Figure 8. Load variation with 1000 tasks.

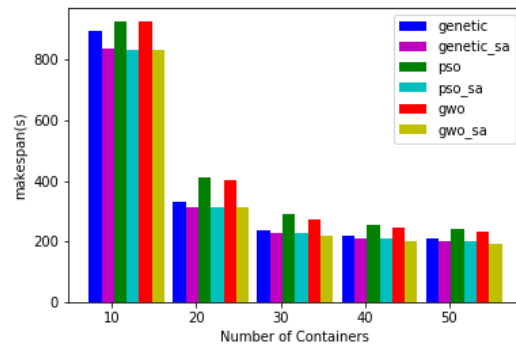


Figure 9. Makespan with 1000 tasks.

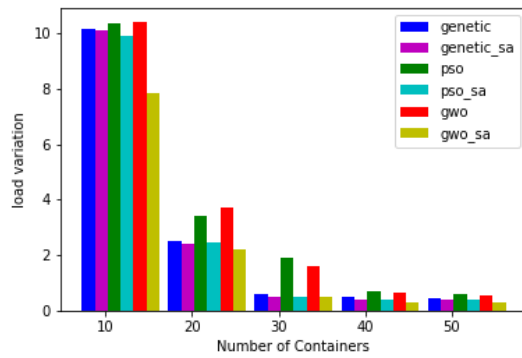


Figure 10. Load variation with 2000 tasks.

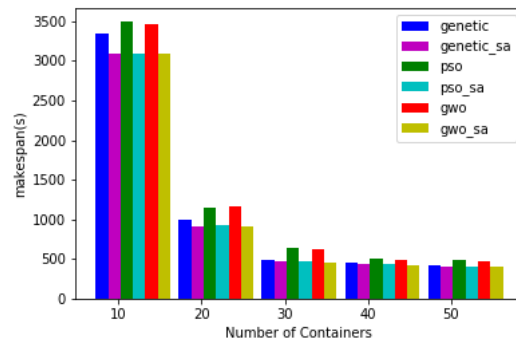


Figure 11. Makespan with 2000 tasks.

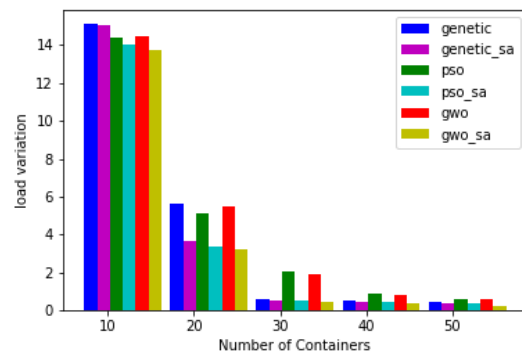


Figure 12. Load variation with 3000 tasks.

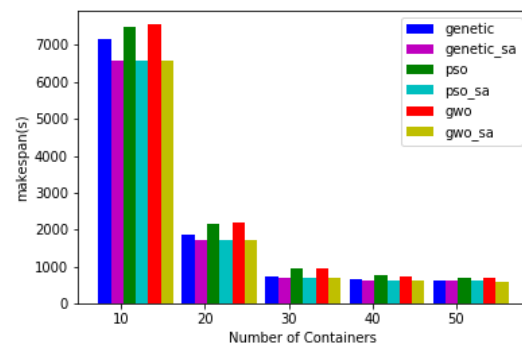


Figure 13. Makespan with 3000 tasks.

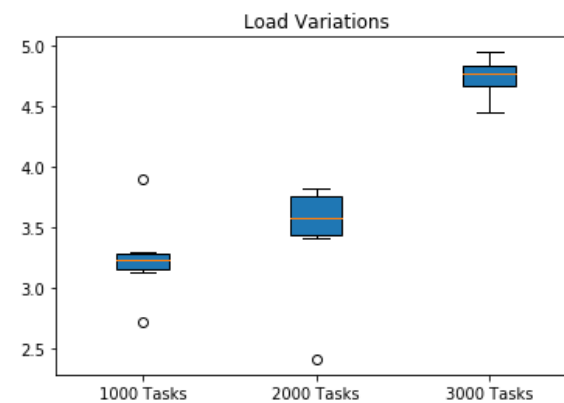


Figure 14. Workload Variations.

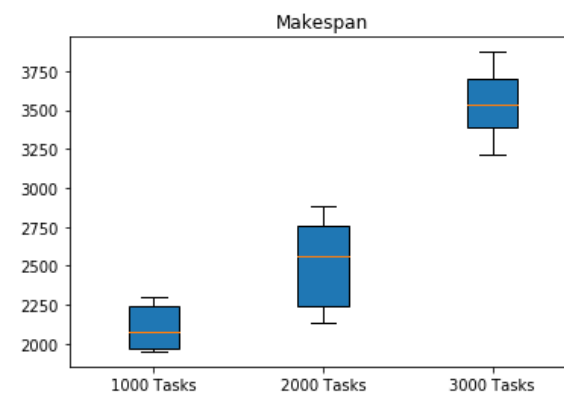


Figure 15. Makespan Variations.

6. Conclusions and Future Directions

The main focus of the work presented in this paper is to distribute the workload among all servers efficiently and minimize the makespan. We propose a Grey Wolf Optimization (GWO) based Simulated Annealing approach to counter the problem of load balancing in the containerized cloud. We perform our experiment by taking 1000, 2000, and 3000 tasks and the number of containers were taken from 10 to 50. Then, we compared our results with the Genetic and Particle Swarm Optimization algorithm and evaluated the proposed algorithms by considering the parameter load variation and makespan. The experimental result indicates that, in most cases, more than 97% of the tasks were meeting their deadline and the Grey Wolf Optimization Algorithm with Simulated Annealing (GWO-SA) performs better than all other approaches in terms of load variation and makespan.

For future perspective, we have found several open issues to address in the containerized cloud environment. Firstly, efficient virtual machine sizing plays an important role in resource utilization. In the future, we will extend the work going ahead with virtual

machine sizing in the containerized cloud. Secondly, categorizing tasks into groups and assigning the same VM for a group of tasks with similar requirement can improve the system performance. Thirdly, migrating VM and container can also balance the workload by migrating tasks from overloaded VM to lightly loaded VM. So, migrating VM or container is better and migrating both is another research challenge. The fourth problem is the resource elasticity, in a dynamic cloud environment, there might be the requirement of additional resources during run time. So, dealing with dynamic resource requirements is another research challenge. We will work on these open issues in our future research.

Author Contributions: M.K.P.: conceptualization, methodology, writing—original draft preparation; B.S.: data curation, supervision; A.K.T.: methodology, supervision; S.M.: investigation, validation, reviewing and finalizing the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: We have used Google Cluster Workload Traces 2019 which is publicly available by Google at <https://research.google/tools/datasets/google-cluster-workload-traces-2019/>, accessed on 1 January 2020.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dimitri, N. Pricing cloud IaaS computing services. *J. Cloud Comput.* **2020**, *9*, 14. [CrossRef]
2. Sandholm, T.; Lee, D. Notes on Cloud computing principles. *J. Cloud Comput.* **2014**, *3*, 21. [CrossRef]
3. Azadi, M.; Emrouznejad, A.; Ramezani, F.; Hussain, F.K. Efficiency Measurement of Cloud Service Providers Using Network Data Envelopment Analysis. *IEEE Trans. Cloud Comput.* **2022**, *10*, 348–355. [CrossRef]
4. Elmougy, S.; Sarhan, S.; Joundy, M. A novel hybrid of Shortest job first and round Robin with dynamic variable quantum time task scheduling technique. *J. Cloud Comput.* **2017**, *6*, 12. [CrossRef]
5. Gawali, M.B.; Shinde, S.K. Task scheduling and resource allocation in cloud computing using a heuristic approach. *J. Cloud Comput.* **2018**, *7*, 4. [CrossRef]
6. Lawanya Shri, M.; Ganga Devi, E.; Balusamy, B.; Kadry, S.; Misra, S.; Odusami, M. A fuzzy based hybrid firefly optimization technique for load balancing in cloud datacenters. In Proceedings of the International Conference on Innovations in Bio-Inspired Computing and Applications, Kochi, India, 17–19 December 2018; Springer: Cham, Switzerland, 2018; pp. 463–473.
7. Zhou, Z.; Abawajy, J.; Chowdhury, M.; Hu, Z.; Li, K.; Cheng, H.; Alelaiwi, A.A.; Li, F. Minimizing SLA violation and power consumption in Cloud data centers using adaptive energy-aware algorithms. *Future Gener. Comput. Syst.* **2018**, *86*, 836–850. [CrossRef]
8. Ibrahim, A.A.Z.A.; Kliazovich, D.; Bouvry, P. Service level agreement assurance between cloud services providers and cloud customers. In Proceedings of the 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, Colombia, 16–19 May 2016; pp. 588–591.
9. Kishor, A.; Niyogi, R.; Chronopoulos, A.; Zomaya, A. Latency and Energy-Aware Load Balancing in Cloud Data Centers: A Bargaining Game Based Approach. *IEEE Trans. Cloud Comput.* **2021**. [CrossRef]
10. Shahid, M.A.; Islam, N.; Alam, M.M.; Su’ud, M.M.; Musa, S. A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach. *IEEE Access* **2020**, *8*, 130500–130526. [CrossRef]
11. Kazeem Moses, A.; Joseph Bamidele, A.; Roseline Oluwaseun, O.; Misra, S.; Abidemi Emmanuel, A. Applicability of MMRR load balancing algorithm in cloud computing. *Int. J. Comput. Math. Comput. Syst. Theory* **2021**, *6*, 7–20. [CrossRef]
12. Pantazoglou, M.; Tzortzakis, G.; Delis, A. Decentralized and Energy-Efficient Workload Management in Enterprise Clouds. *IEEE Trans. Cloud Comput.* **2016**, *4*, 196–209. [CrossRef]
13. Shafiq, D.A.; Jhanjhi, N.; Abdullah, A. Load balancing techniques in cloud computing environment: A review. *J. King Saud Univ.-Comput. Inf. Sci.* **2021**, *34*, 3910–3933. [CrossRef]
14. Tong, Z.; Deng, X.; Ye, F.; Basodi, S.; Xiao, X.; Pan, Y. Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. *Inf. Sci.* **2020**, *537*, 116–131. [CrossRef]
15. Nawrocki, P.; Grzywacz, M.; Sniezynski, B. Adaptive resource planning for cloud-based services using machine learning. *J. Parallel Distrib. Comput.* **2021**, *152*, 88–97. [CrossRef]
16. Yin, L.; Li, P.; Luo, J. Smart contract service migration mechanism based on container in edge computing. *J. Parallel Distrib. Comput.* **2021**, *152*, 157–166. [CrossRef]
17. 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]
18. Piraghaj, S.F.; Dastjerdi, A.V.; Calheiros, R.N.; Buyya, R. ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers. *Softw. Pract. Exp.* **2017**, *47*, 505–521. [CrossRef]

19. Maenhaut, P.J.; Volckaert, B.; Ongenaes, V.; De Turck, F. Resource management in a containerized cloud: Status and challenges. *J. Netw. Syst. Manag.* **2020**, *28*, 197–246. [[CrossRef](#)]
20. Souppaya, M.; Morello, J.; Scarfone, K. *Application Container Security Guide*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2017.
21. Rana, N.; Abd Latiff, M.S.; Abdulhamid, S.M.; Misra, S. A hybrid whale optimization algorithm with differential evolution optimization for multi-objective virtual machine scheduling in cloud computing. *Eng. Optim.* **2021**, 1–18. [[CrossRef](#)]
22. Lu, J.; Zhao, W.; Zhu, H.; Li, J.; Cheng, Z.; Xiao, G. Optimal machine placement based on improved genetic algorithm in cloud computing. *J. Supercomput.* **2022**, *78*, 3448–3476. [[CrossRef](#)]
23. Wang, T.; Liu, Z.; Chen, Y.; Xu, Y.; Dai, X. Load balancing task scheduling based on genetic algorithm in cloud computing. In Proceedings of the 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, Dalian, China, 24–27 August 2014; pp. 146–152.
24. Verma, S.; Sood, N.; Sharma, A.K. Genetic algorithm-based optimized cluster head selection for single and multiple data sinks in heterogeneous wireless sensor network. *Appl. Soft Comput.* **2019**, *85*, 105788. [[CrossRef](#)]
25. Devaraj, D.; Banu, R.N. Genetic algorithm-based optimisation of load-balanced routing for AMI with wireless mesh networks. *Appl. Soft Comput.* **2019**, *74*, 122–132.
26. Wang, B.; Li, J. Load balancing task scheduling based on multi-population genetic algorithm in cloud computing. In Proceedings of the 2016 35th Chinese Control Conference (CCC), Chengdu, China, 27–29 July 2016; pp. 5261–5266.
27. Babbar, H.; Parthiban, S.; Radhakrishnan, G.; Rani, S. A genetic load balancing algorithm to improve the QoS metrics for software defined networking for multimedia applications. *Multimed. Tools Appl.* **2022**, *81*, 9111–9129. [[CrossRef](#)]
28. Hussain, A.; Manikanthan, S.; Padmapriya, T.; Nagalingam, M. Genetic algorithm based adaptive offloading for improving IoT device communication efficiency. *Wirel. Netw.* **2020**, *26*, 2329–2338. [[CrossRef](#)]
29. Dam, S.; Mandal, G.; Dasgupta, K.; Dutta, P. Genetic algorithm and gravitational emulation based hybrid load balancing strategy in cloud computing. In Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT), Hooghly, India, 7–8 February 2015; pp. 1–7.
30. Dasgupta, K.; Mandal, B.; Dutta, P.; Mandal, J.K.; Dam, S. A genetic algorithm (ga) based load balancing strategy for cloud computing. *Procedia Technol.* **2013**, *10*, 340–347. [[CrossRef](#)]
31. Pilavare, M.S.; Desai, A. A novel approach towards improving performance of load balancing using Genetic Algorithm in cloud computing. In Proceedings of the 2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), Coimbatore, India, 19–20 March 2015; pp. 1–4.
32. Pradhan, A.; Bisoy, S.K. A novel load balancing technique for cloud computing platform based on PSO. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 3988–3995. [[CrossRef](#)]
33. Agarwal, R.; Baghel, N.; Khan, M.A. Load Balancing in Cloud Computing using Mutation Based Particle Swarm Optimization. In Proceedings of the 2020 International Conference on Contemporary Computing and Applications (IC3A), Lucknow, India, 5–7 February 2020; pp. 191–195.
34. Devaraj, A.F.S.; Elhoseny, M.; Dhanasekaran, S.; Lydia, E.L.; Shankar, K. Hybridization of firefly and Improved Multi-Objective Particle Swarm Optimization algorithm for energy efficient load balancing in Cloud Computing environments. *J. Parallel Distrib. Comput.* **2020**, *142*, 36–45. [[CrossRef](#)]
35. Ramezani, F.; Lu, J.; Hussain, F.K. Task-based system load balancing in cloud computing using particle swarm optimization. *Int. J. Parallel Program.* **2014**, *42*, 739–754. [[CrossRef](#)]
36. Alguliyev, R.M.; Imamverdiyev, Y.N.; Abdullayeva, F.J. PSO-based load balancing method in cloud computing. *Autom. Control Comput. Sci.* **2019**, *53*, 45–55. [[CrossRef](#)]
37. Ebadifard, F.; Babamir, S.M. A PSO-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment. *Concurr. Comput. Pract. Exp.* **2018**, *30*, e4368. [[CrossRef](#)]
38. Liu, Z.; Wang, X. A PSO-based algorithm for load balancing in virtual machines of cloud computing environment. In Proceedings of the International Conference in Swarm Intelligence, Shenzhen, China, 17–20 June 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 142–147.
39. Miao, Z.; Yong, P.; Mei, Y.; Quanjun, Y.; Xu, X. A discrete PSO-based static load balancing algorithm for distributed simulations in a cloud environment. *Future Gener. Comput. Syst.* **2021**, *115*, 497–516. [[CrossRef](#)]
40. Sefati, S.; Mousavinasab, M.; Zareh Farkhady, R. Load balancing in cloud computing environment using the Grey wolf optimization algorithm based on the reliability: Performance evaluation. *J. Supercomput.* **2022**, *78*, 18–42. [[CrossRef](#)]
41. Patel, D.; Patra, M.K.; Sahoo, B. GWO Based task allocation for load balancing in containerized cloud. In Proceedings of the 2020 International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 26–28 February 2020; pp. 655–659.
42. Abed-Alguni, B.H.; Alawad, N.A. Distributed Grey Wolf Optimizer for scheduling of workflow applications in cloud environments. *Appl. Soft Comput.* **2021**, *102*, 107113. [[CrossRef](#)]
43. Yuvaraj, N.; Karthikeyan, T.; Pragmaash, K. An improved task allocation scheme in serverless computing using gray wolf Optimization (GWO) based reinforcement learning (RL) approach. *Wirel. Pers. Commun.* **2021**, *117*, 2403–2421. [[CrossRef](#)]
44. Lipare, A.; Edla, D.R.; Kuppili, V. Energy efficient load balancing approach for avoiding energy hole problem in WSN using Grey Wolf Optimizer with novel fitness function. *Appl. Soft Comput.* **2019**, *84*, 105706. [[CrossRef](#)]

45. 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](#)]
46. Sahoo, S.; Sahoo, B.; Turuk, A.K. A Learning Automata-Based Scheduling for Deadline Sensitive Task in The Cloud. *IEEE Trans. Serv. Comput.* **2021**, *14*, 1662–1674. [[CrossRef](#)]
47. Xie, Y.; Jin, M.; Zou, Z.; Xu, G.; Feng, D.; Liu, W.; Long, D. Real-Time Prediction of Docker Container Resource Load Based on a Hybrid Model of ARIMA and Triple Exponential Smoothing. *IEEE Trans. Cloud Comput.* **2022**, *10*, 1386–1401. [[CrossRef](#)]
48. Zou, Z.; Xie, Y.; Huang, K.; Xu, G.; Feng, D.; Long, D. A Docker Container Anomaly Monitoring System Based on Optimized Isolation Forest. *IEEE Trans. Cloud Comput.* **2022**, *10*, 134–145. [[CrossRef](#)]
49. Makasarwala, H.A.; Hazari, P. Using genetic algorithm for load balancing in cloud computing. In Proceedings of the 2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Ploiesti, Romania, 30 June–2 July 2016; pp. 1–6. [[CrossRef](#)]
50. Dave, A.; Patel, B.; Bhatt, G.; Vora, Y. Load balancing in cloud computing using particle swarm optimization on Xen Server. In Proceedings of the 2017 Nirma University International Conference on Engineering (NUiCONE), Ahmedabad, India, 23–25 November 2017; pp. 1–6. [[CrossRef](#)]