Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

DSTS: A hybrid optimal and deep learning for dynamic scalable task scheduling on container cloud environment

Saravanan Muniswamy* and Radhakrishnan Vignesh

Abstract

Containers have grown into the most dependable and lightweight virtualization platform for delivering cloud services, offering flexible sorting, portability, and scalability. In cloud container services, planner components play a critical role. This enhances cloud resource workloads and diversity performance while lowering costs. We present hybrid optimum and deep learning approach for dynamic scalable task scheduling (DSTS) in container cloud environment in this research. To expand containers virtual resources, we first offer a modified multi-swarm coyote optimization (MMCO) method, which improves customer service level agreements. Then, to assure priority-based scheduling, we create a modified pigeon-inspired optimization (MPIO) method for task clustering and a rapid adaptive feedback recurrent neural network (FARNN) for pre-virtual CPU allocation. Meanwhile, the task load monitoring system is built on a deep convolutional neural network (DCNN), which allows for dynamic priority-based scheduling. Finally, the presentation of the planned DSTS methodology will be estimated utilizing various test vectors, and the results will be associated to present state-of-the-art techniques.

Keywords: Cloud container, Task scheduling, Virtual resources, Task clustering, Priority based scheduling, Load monitoring

Introduction

Cloud computing, which provides the computer services required for the Internet, has become one of the most popular technologies for the economy, society, and people in latest years [1]. Due to the recent growth in the load of different and sophisticated clouds like the Internet of Things (IoT) devices, machine learning programmes, coursing A/V services, and cloud memory, mandate for several cloud amenities has risen substantially [2]. With the introduction of numerous virtualization technologies like as VMware, Citrix, KVM, and Zen [3], the cloud computing business has evolved fast in recent years. Despite their widespread use, virtualization technologies have a number of drawbacks, including high time consumption, extended runs and shutdowns, and difficult planning and migration procedures [4]. The hardware is virtualized in the conventional setup, and each virtual machine running the whole operating system supervises the computer's application activities [5]. The application process in the container communicates directly with the host kernel, but the container does not have its own kernel or hardware virtualization. Containers are therefore far lighter than typical virtual computers [6, 7].

Furthermore, the spread of microservices, self-driving vehicles, and smart infrastructure is predicted to boost cloud service growth [8]. The backbone of cloud computing is virtualization technology, which enables applications to be detached from fundamental infrastructure by sharing resources and executing various programmes independently [9]. Containers have grown in popularity as a novel virtualization approach in recent years, bringing conventional fundamental machines (VMs) to numerous

*Correspondence: saravananm@presidencyuniversity.in

Department of Computer Science and Engineering, School of Engineering, Presidency University, Bengaluru, Karnataka 560064, India

auspicious characteristics including united host operating systems, quicker boot times, portability, scalability, and faster deployment [10]. Containers allow apps to store all of their dependencies in the sandbox, allowing them to construct autonomous working hours from the platform while also increasing productivity and portability [11]. Dockers, LXC, and Kubernetes are just a few of the container technologies available. Furthermore, several cloud service providers run containers on virtual machines (VMs) to increase container seclusion, performance, and system management [12, 13]. Container technology is gaining traction among developers, and it's now being used to deploy a wide range of microservices and applications, including smart devices, IoT, and fog / edge computing [14]. As a consequence, to fulfil the increased demand, numerous cloud service suppliers have begun to provide container-based cloud services. Google Container Engine, Amazon Re-Container Service, and Azure Container Service are other examples. The cloud computing paradigm is being revolutionised by container technology [15]. Running containerized applications, in the eyes of the cloud service provider, produces a compression layer that deals with cluster management. The primary container orchestration sites in the base cluster for automating, measuring, and controlling container-based infrastructure are Docker Swarm and Google Kubernetes [16, 17]. A container cluster's overall structure comprises of management nodes and task nodes. The cluster and container node work nodes, on the other hand, are the responsibility of the management nodes [18]. In addition, the manager keeps track of the cluster's location by verifying the node's position on a regular basis. The planning components, which are responsible for spreading loads among cluster nodes and controlling the container life process [19], play a precarious part in container transposition. Depending on the technology, container planning may take many different shapes. As a result, the primary goal of container planning is to get the containers started on the ideal host and link them together [20].

Our contributions

A dynamic scalable task scheduling (DSTS) approach is offered for cloud container environments as a way to improve things even further. The main contributions of our proposed DSTS approach are given as follows:

1. To provide a dynamic scalable task scheduling system for container cloud environments in order to reduce the make span while using less computing resources and containers than current algorithms.
2. To offer a unique clustered priority-based task scheduling technique that improves the scheduling system's flexibility to cloud environment while also speeding convergence.
3. Create a task load monitoring system that allows for dynamic scheduling depending on priority.
4. Using various test scenarios and metrics, assess the performance of the suggested dynamic scalable task scheduling.

The balance of the paper is placed as proceeds: The second segment summarises recent work on job scheduling for cloud containers. We go through the issue technique and system design in Problem methodology and system design section. The suggested dynamic scalable task scheduling (DSTS) model's functioning function is designated in Proposed methodology section. Simulation results and analysis section deliberates the simulation findings and comparison analyses. Finally, Conclusion section brings the paper to a close.

Related works

Many studies for scalable task scheduling for cloud containers have been suggested in recent years all around the globe. Table 1 summarises and tabulates the literature with research gaps in many categories.

Zhao et al. [21] studied to improve today's cloud services by reviewing the workings of projects for planning next-generation containers. In particular, this work creates and analyzes a new model that respects both workload balance and performance. Unlike previous studies, the model uses statistical techniques to create confusion between load balance and utility performance in a single optimization problem and solve it effectively. The difficult element is that certain sub-issues are more complicated, necessitating the use of heuristic guidance. Liu et al. [22] suggested a multi-objective container scheduling technique based on CPU node consumption, memory usage across all nodes, time to transport pictures over the network, container-node connections, and container clustering, all of which impact container programme performance. The author provides the metric techniques for all the important components, sets the relevant qualifying functions, and then combines them in order to pick the suitable nodes for the layout of the containers to be allotted in the planning process. Lin et al. [23] suggested a multi-objective optimization model for container-based micro service planning that uses an ant colony method to tackle the issue. The method takes into account not only the physics nodes' use of computer and storing possessions, but also the numeral of multi-objective requirements and the loss rate of physics nodes. These approaches make use of prospective algorithms' quality assessment skills to assure the correctness of pheromone updates and to increase the likelihood of utilising multi-functional horistic information to choose the optimum route. Adhikari et al. [24] suggested an energy-efficient container-based scheduling (EECS) technique for fast

Table 1 Summary of research gaps

Ref	Proposed	Methodology	Parameters	Future work
[21]	Diego	Heuristic algorithms	Execution time	The prototypes described were extending to wider environment; integrated into planned cloud services.
[22]	Multiopt	Virtual machine	Response time	To move containers without affecting or reducing the use of cloud services.
[23]	MOO-ACA	GA_MOCA algorithm	Network transmission overhead	Use scheduling methods in cloud containers to reduce the problem of algorithm time.
[24]	EECS	APSO	Temperature	Create a cloud environment for IoT applications that is dynamic and container-based, and allocate apps to the most appropriate containers.
[25]	Container-based virtualized model	VM	Execution time	Analyze the impact of post-failure work restructuring, interruptions due to work proximity in multiple cloud environments
[26]	Adaptive fair-share method	GPU memory allocation algorithm	GPU memory utilization	Improved Tensor Flow multi-container processing allows to securely share a GPU
[27]	ECSched	MCFP algorithm	Fraction of containers	To embrace more intricate circumstances, consider container dependencies and resource dynamics in the scheduler.
[28]	SRPSM	VM	Sensitivity	Searching multiple containers on same VM to perform multiple tasks in parallel
[29]	KCSS	Machine learning	Computing time	KCSS to identify residential containers and improve global performance.
[30]	CANSS	Naive Bayes	Cache hit ratio	Use artificial intelligence algorithms to compute if cache localization can be achieved
[31]	State-of-the art scheduling algorithm	Optimization algorithm	Throughput	Create a security alert table to avoid security issues related to the use of containers in your cloud infrastructure.
[32]	Skippy scheduling container	MCDM algorithm	function execution time	By implementing high-level operational goals, customize key planning parameters to explore specific aspects.

inheritance of various IoT and non-IoT chores. To determine the optimum container for each work, an accelerated particle swarm optimization (APSO) method with minimum latency is applied. Another significant duty in the cloud environment is resource planning in order to make the greatest use of resources on cloud servers. Ranjan et al. [25] shown how to design energy-efficient operations in program-limited data centres using container-based virtualization. Policies Containers provide users the freedom to get vital resources that are suited to their own need.

Chen et al. [26] suggested a functional restructuring system to control the operating sequence of each container in order to achieve maximum performance gain, as well as an adaptive fair-sharing system to effectively share the container-based virtualized environment. They also suggested a checkpoint-based system, which would be particularly useful for load balancing. Hu et al. [27] suggested the ECSched improved container scheduler for planning simultaneous requests over several clusters with varied resource restrictions. Define a container planning issue as a minimal cost flow (MCFP) problem and communicate container needs utilizing a specialised graphical data format. ECSched allows you to design a flow network based on a set of needs while also allowing MCFP algorithms to plan fixed requests live. Evaluate ECSched in a variety of test clusters and run large-scale planning overhead simulations to see how it performs. Experiments demonstrate that ECSched is superior at container planning in terms of container function and resource performance, and that large clusters only introduce minor and acceptable planning overlays.

For the VAS operating system, Rajasekar et al. [28] provided a planning and resource strategy. Infrastructure (IaaS) suppliers provide computer, networking, and storage services. As a result, the VAS design may effectively plan this burden at important periods utilising a range of features and quality of service (QoS). The method is scalable and dynamic, altering the load and base as needed. KCSS is a Kubernetes Container Scheduling Strategy introduced by Menouer et al. [29]. To satisfy the demands of Maxpania and Cloud providers, KCSS intends to optimise the scheduling of many containers that users submit to the Internet in order to increase customer performance based on energy usage. Due to the table's cloud infrastructure level and restricted perspective of user demands, single-based planning is less efficient. KCSS is responsible for introducing multi-criterion node selection. A cache-aware scheduling approach based on neighbourhood search was suggested by Li et al. [30]. Job categorization, node resource allocation, node clustering, and cache target planning are the four sub-issues of this paradigm. It's separated into three sorts, and then various resources are transferred to the node depending on how well it performs. The work is stored late after the nodes with comparable functions are assembled. Ahmad et al. [31] looked at a variety of current container planning approaches in order to continue their study in this hot topic. The research is based on mathematical modelling, heuristics, Meta heuristics, and machine learning, and it divides planning approaches into four groups depend upon the algorithm of optimization used to construct the map. Formerly, based on performance measurements, examine and identify important benefits and difficulties for each class of planning approach, as well as main hardware issues. Finally, this study discusses how successful research might improve the future potential of innovative container technologies. The container planning strategy provided by Rausch et al. [32] helps to make good use of the margin infrastructure on these sites. They'll also illustrate how to modify the weight of scheduling controls automatically to optimise high-level performance objectives like task execution time, connection use, and cloud performance costs. Implement a Kubernetes container orchestration system prototype and install bridges on the edges where it was constructed. Utilizing hints given by the test's frequent loads, evaluate the system using micro-organized simulations in different infrastructure situations.

Problem methodology and system design

Problem statement

- Learning automata are used to suggest a self-accommodating duty scheduling algorithm (ADATSA) [33]. In conjunction through the futile formal of resources and the in succession stage of responsibilities in the present surroundings, the algorithm efficiently leveraged the re-enforcement educating capacity of learning mechanisms and achieves an operative remuneration-fine system for arranging activities. A charge load observing framework for actual-time observing of the surrounding and planning assessment opinion, as well as the establishment of a buffer queue for priority scheduling. To compare the non-automata technology-based algorithm PSOS, the ADATSA algorithm to learning automata-based algorithm LAEAS, and the K8S planning engine relating resource imbalance, resource residual degree, and QoS, researchers used the Kubernetes platform to pretend various planning circumstances.
- In general, cloud computing environments need great portability, and containerisation assures surroundings compatibility by en-capsulation uses collected with their libraries, configuration files, and other needs, allowing consumers [34] to quickly migrate and set up programmes across gatherings.
- However, there are still certain obstacles to be solved in this project. Furthermore, the study literature [21–33, 35] lacks methods and models that enable dynamic scalability, in which consumers get QoS and good performance [36] while using the fewest amount of cloud resources possible, particularly for containerized services hosted on the cloud.
- Cloud computing services benefit from dynamic scalability, which provides on-demand, timely, and dynamically changeable computing resources.
- However, since the container cloud environment is very changeable and unpredictable, the environment exemplary derived as of static reward-penalty components might not be optimum. ADATSA algorithm does not take into account diversity of cloud resources. Users' demands for cloud resources are often diverse, and operator responsibilities are typically completed by a combination of heterogeneous cloud services.

According to above gathered research gaps it needs proposed methodology. Hybrid optimal and deep learning is proposed for dynamic scalable task scheduling (DSTS). The main contributions are list as follows:

- A modified multi-swarm coyote optimization (MMCO) algorithm is used for scaling the containers virtual resources which enhance customer service level agreements.
- A modified pigeon-inspired optimization (MPIO) algorithm is proposed for task clustering and the fast adaptive feedback recurrent neural network (FARNN) is used for pre-virtual CPU allocation to ensure priority based scheduling.

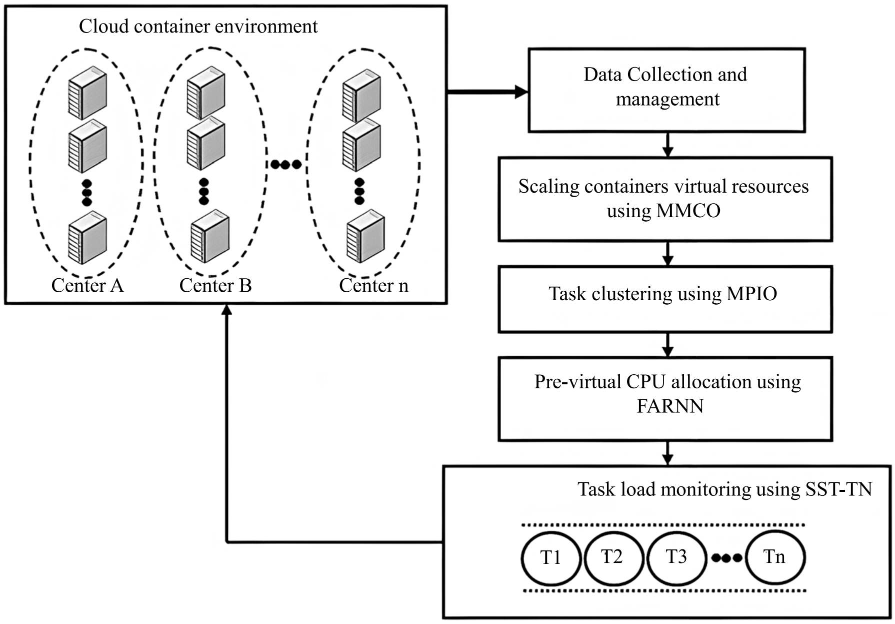

Cloud container environment
Center A
Center B
Center n
Data Collection and management
Scaling containers virtual resources using MMCO
Task clustering using MPIO
Pre-virtual CPU allocation using FARNN
Task load monitoring using SST-TN
T1
T2
T3
Tn

Fig. 1 Dynamic scalable task scheduling (DSTS) model

- The task load monitoring mechanism is designed based on deep convolutional neural network (DCNN) which achieves dynamic scheduling based on priority.

System design of proposed methodology

Before being deployed to the cloud, programmes must be imaged and encased in the container cloud podium. The purpose of charge planning is to assign container illustrations to the most appropriate node in order to create the most effective utilization of accessible means. The difficulty of mapping relationships between containers and nodes may be represented as task scheduling in container cloud. Figure 1 depicts the system architecture of the proposed dynamic scalable task scheduling (DSTS) paradigm. The DSTS model includes a number of processes, including container virtual resource scaling, task clustering, pre-virtual CPU allocation, and task load monitoring.

Proposed methodology

In this section, we describe the following process such as containers virtual resources scaling, task clustering, pre-virtual CPU allocation and task load monitoring mechanism.

Container virtual resources scaling using MMCO algorithm

The goal of cloud service level agreements (SLAs) is for service providers to have a common understanding of priority areas, duties, warranties, and service providers. It specifies the dimensions and duties of the parties participating in the cloud setup, as well as the timeframe for reporting or resolving system vulnerabilities. As more firms depend on external suppliers for their vital systems, programmes, and data, service level agreements are becoming more important. The Cloud SLA assures that cloud providers satisfy specific enterprise-level criteria and provide clients a clear distribution. If the provider fails to satisfy the requirements of the guarantee, it may be subject to financial penalties such as service time credit. The modified multi-swarm coyote optimization (MMCO) method was used to scale virtual resources in containers, improving customer service level agreements. MMCO coyote population is split into two groups F_d consists of F_q each coyote; the number of coyotes in each pack is constant and consistent across all packs in the first suggestion. As a result, multiplying the algorithm's total population gives algorithm's entire population $F_d \in F^*$ and $F_q \in F^*$.Furthermore, the social position of the people q^{th} coyote from the woods d^{th} cram everything in a^{th} the current time has been specified.

$$SOC_q^{d.a} = \overrightarrow{b} = (b_1, b_2, ..b_h) \tag{1}$$

where C demonstrates the number of elements that go into making a choice, It also means that the coyote has adapted to its environment $FIT_q^{d.a} \in J$. Establishing the social position of the people q^{th} coyote from the woods d^{th} a compilation of p^{th} the dimension is specified via a vector.

$$SOC_{d.p}^{q.a} = Ua + j_p.\left(na_p - Ua_p\right) \tag{2}$$

where Ua_p and na_p stands for, respectively, the bottom and top limits of the range p^{th} choice variable and j_p is a true random number created inside the range's bounds [0, 1] Using a probability distribution that is uniform in nature.

To determine the fitness function of each coyote, $F_q \times F_d$ Coyotes in the environment, depending their socioeconomic situations

$$FIT_q^{d.t} = m\left(SOC_q^{d.a}\right) \tag{3}$$

In the case of a minimization problem, the solution's Alpha d^{th} crams everything in a^{th} a split second in time

$$Alpha^{d.A} = \left\{SOC_q^{\backslash d.A} \middle| \arg_{q=\{1,2....f_d\}} \min l\left(SOC_q^{d.A}\right)\right\} \tag{4}$$

MMCO integrates all of the coyote's information and calculates the cultural propensity of each pack:

$$Cul_p^{d.A} = \begin{cases} z^{d.A}_{\frac{(F_T+1)}{2}.i} & F_d \ is \ odd \\ \dfrac{z^{d.A}_{\frac{Ft}{2}.i} + z^{d.A}_{\left(\frac{F_t}{2}+1\right).p}}{2} .otherwise \end{cases} \tag{5}$$

where Z_D, the social standing of all coyotes in the region is indicated by the letter A. d^{th} in a hurry A^{th} p in the price range at the given point in time [1, C]. At the same time, the Alpha has an effect on coyotes (δ_1) and by the other coyotes in the pack (δ_2),

$$\delta_1 = Alpha^{d.A} - SOC_{qj_1}^{d.A} \tag{6}$$

$$\delta_2 = Cult^{d.A} - SOC_{qj_2}^{d.A} \tag{7}$$

The alpha δ_1 Influence distinguishes a coyote from the rest of the pack in terms of culture, Qj_1, to the coyote leader, whereas the pack's clout δ_2, shows a cultural distinction from a random coyote Qj_2, to the cultural tendencies of the pack. In MMCO algorithm, during the initialization of the method, the swarm, also known as stands, is randomly seeded to the search space.

$$a_{s.p} = U_p + j_{s.p} \times \left(X_p - U_p\right) \tag{8}$$

where, $a_{s.\,p}$ represents s^{th} a hive of activity p^{th} dimension, U_p and X_p are the bottom and top edges of the solution space, respectively, and $_{s,\,p}$ is a range of uniformly generated random numbers [0, 1].

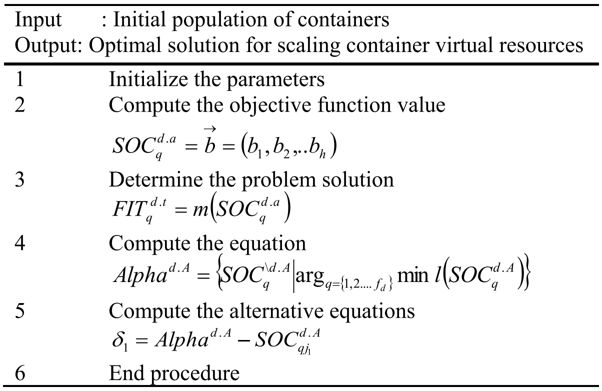
$$T = \arg\min\left\{l\left(\overrightarrow{a}\right)\right\} \tag{9}$$

To generate Multi swarm from this point, two different equations may be used.

$$K_{A.p} = a_{s.p} + \alpha \times \left(T_p - a_{o.p}\right) \tag{10}$$

$$K_{A.p} = a_{s.p} + \alpha \times \left(a_{s.p} - a_{o.p}\right) \tag{11}$$

where, sindices must not be identical and α factor of scalability. The equation used to update the dimension of a swarm that will be formed for a Swarm is an important part of the process. The working function of the process of container virtual resources scaling is given in Algorithm 1.



```
Input  : Initial population of containers
Output: Optimal solution for scaling container virtual resources
Initialize the parameters
Compute the objective function value
    $SOC_q^{d.a} = \vec{b} = (b_1, b_2,..b_h)$
Determine the problem solution
    $FIT_q^{d.t} = m(SOC_q^{d.a})$
Compute the equation
    $Alpha^{d.A} = \{SOC_q^{\backslash d.A} | \arg_{q=\{1,2....f_d\}} \min l(SOC_q^{d.A})\}$
Compute the alternative equations
    $\delta_1 = Alpha^{d.A} - SOC_{qj_1}^{d.A}$
End procedure
```

Algorithm 1 Container virtual resources scaling using MMCO algorithm

Task clustering using modified pigeon-inspired optimization (MPIO) algorithm

Clustering is a procedure that divides tasks into different categories depending on increasing application demand, such as load balancing clusters, high availability clusters, and compute clusters. The primary emphasis of load balancing clusters is resource use on the host system, particularly the virtual machine. These clusters are utilised to balance constant and dynamic loads, as well as to move the application from one cloud provider to the next. The second kind is fault-tolerant high-availability clusters that are built for tip failure. For task clustering, we used a modified pigeon-inspired optimization (MPIO) algorithm. The activation function ties the information about the concealed state of prior deadlines to the item in the current chronology, and it provides it to the entrance gate as follows:

$$H_r = \upsilon\left(X_r K^H + t_{r-1} \nu^H + b_H\right) \tag{12}$$

where E_S is recall gate. X_r is input at each time step s and T_{S-1} represent the previous time step's hidden state

T − 1. Z^e is the input layer's heaviness and v^e is recurring heaviness of the concealed state. The b_e is the bias of the input layer. The following are the equations for the two tasks:

$$i_r = \upsilon\left(X_r K^i + t_{r-1} v^i + b_i\right) \tag{13}$$

$$\widetilde{E}_s = \tanh\left(X_r Z^e + t_{r-1} v^e + b_e\right) \tag{14}$$

$$E_r = E_{r-1}{}^* H_r + i_r{}^* \widetilde{E}_s \tag{15}$$

The hidden levels at which the sigmoid activation function is anticipated are determined by the output gate. To create a create output, sends to the newly changed cell level function and multiplies as follows.

$$Z_r = \upsilon\left(X_r X^Z + t_{r-1} v^Z + b_Z\right) \tag{16}$$

$$t_r = Z_r{}^* \tanh\left(E_r\right) \tag{17}$$

The update gateway functions similarly to a forget-me-not and LSTM input gateway. The weight is multiplied by the current input, and the weight is multiplied by the level hidden at the prior time point. Using the sigmoid function to find the values of one from zero and one, the contributions of the two possibilities are merged

$$L_r = \upsilon\left(X_r X^L + d_{r-1} v^l + b_l\right) \tag{18}$$

where W_S symbolize the gate for updating, the Y_S at a given time step, the input vector s while c_{S-1} is the earlier output from preceding entities. The K^s is the mass of the input layer, and u^W is the repeated mass. The b_s is the bias of the input layer. The reset gate's output is as follows:

$$s_r = \upsilon\left(X_r K^s + t_{r-1} v^S + b_S\right) \tag{19}$$

The reset gate is employed in the new memory phone to accumulate the in sequence of the preceding phase. The network will be able to choose just relevant earlier events in chronological sequence as a result of this. The present memory contact is as follows:

$$\widetilde{E}_r = \tanh\left(X_r K + v(s_r \Theta d_{r-1})\right) \tag{20}$$

$$d_r = L_r \Theta d_{r-1} + (1 - L_r) \Theta \upsilon\left(\widetilde{E}_r\right) + b_d \tag{21}$$

Each pigeon has a specific scenario when it comes to the optimization challenge.

$$X_i = [x_{i1}, x_{i2}, \ldots x_{ic}] \tag{22}$$

where c is the scope of the problem to be tackled1, 2... M, M is the pigeons' population; each pigeon has a velocity that is stated as follows:

$$u_i = [U_{i1}, U_{i2}, \ldots U_{im}] \tag{23}$$

First, figure out where the dust is in the search region and how fast it is moving. Then, as the number of repetitions grows, so does the difficulty, the u_i can be updated by repeating the following steps

$$u_i(r) = u_i(r-1).e^{-sr} + Rand.(X_{FBest} - X_i(r-1)) \tag{24}$$

where S is the number of current iterations. Then the next x_i is calculated as follows

$$x_i(r) = x_i(r-1) + u_i(r) \tag{25}$$

```
Input   : no. of tasks, node list, node resource group, target task
Output  : cluster formation
1       Initialize the parameters
2       Compute the tasks using
        $E_r = E_{r-1} * H_r + i_r * \widetilde{E}_s$
3       Determine the sigmoid function using
        $L_r = \upsilon(X_r X^L + d_{r-1} v^l + b_l)$
4       Update the position using
        $X_i(r) = X_i(r-1) + Rand.(X_{Center}(r-1) - X_i(r-1))$
5       Compute the function using
        $m_q(r) = ceil\left(\frac{m_p(r-1)}{2}\right)$
6       Compute the fitness using
        $fitness(X_i(r)) = \frac{1}{H_{Min}(X_i(r)) + \varepsilon}$
```

Algorithm 2 Task clustering using MPIO algorithm

As a result, the iteration position M_{th} can be updated by

$$X_i(r) = X_i(r-1) + Rand.\left(X_{Center}(r-1) - X_i(r-1)\right) \tag{26}$$

$$X_{Center}(r) = \frac{\sum_{i=1}^{m} X_i(r).fitness(X_i(r))}{m_p \sum_{i=1}^{m} fitness\left((X_i(r)\right)} \tag{27}$$

$$m_q(r) = ceil\left(\frac{m_p(r-1)}{2}\right) \tag{28}$$

where H is the present number of the iteration $H = 1$, 2. ...H_{Max}, is the amount of iterations in which the signpost operator is active. The meaning of fitness is to be optimized:

$$fitness\left(X_j(r)\right) = H_{Max}\left(X_j(r)\right) \tag{29}$$

$$fitness(X_i(r)) = \frac{1}{H_{Min}(X_i(r)) + \varepsilon} \tag{30}$$

The pigeon's position will be close to the center point after each iteration which reaches the end R_{Max}. Algorithm 2 describes the operation of the task clustering process utilising the MPIO algorithm.

Pre-virtual CPU allocation using FARNN technique

In cloud computing, the latest virtual processor planning techniques are essential to hide physical resources from running programs and reduce performance during virtualization. However, different QoS requirements for cloud applications make it difficult to evaluate and predict the behavior of virtual processors. Based on the evaluation process, a specific planning plan regulates virtual machine priorities when processing I/O requirements for equitable distribution. Our program evaluates the CPU intensity and I/O intensity of virtual machines, making them very effective in a wide range of tasks. Here we applied fast adaptive feedback recurrent neural network (FARNN) for pre-virtual CPU allocation phase to ensure the priority based scheduling.

The FARNN methodology is a set of computing techniques that use model and method learning to anticipate computer effects by simulating the human brain's problematic-answering process. The three network layers of a normal FARNN approach are the input film, hidden film, and output film. For arrest forecast systems, the input film typically contains the current time interval's recorded MAC address. The following is a format for the MAC address input vector at time T:

$$Y(T) = \left\{y_1, y_2, \ldots, y_j, \ldots, y_l\right\} \tag{31}$$

At the current time, the all MAC address collection is denoted as $Y(T)$. T stands for the overall quantity of MAC addresses in use at any one period. The j^{th} Mac address detection is represented as y_j respectively. The input and network weights are used to compute the hidden layer neutrons.

$$h(T) = Z_1^{t*} Y(T) + a \tag{32}$$

Output film associates the results of the Hidden film and converts them.

$$X(T) = f\left(Z_2^{t*} h(T)\right) = f\left(Z_2^{t*}\left(Z_1^{t*} Y(T) + a\right)\right) \tag{33}$$

The hidden layer output is denoted as $h(T)$ and the output layer output is referred as $X(T)$ respectively. From the Input to Hidden film the weight is denoted as Z_1^t and from the Hidden film to the Output film is stated as Z_2^t respectively. The activation function is indicated as f(.) and the random bias is denoted as an in the output layer. The Feature film is initially combined amongst the Input film and the Hidden film in the rapid adaptive to determine the transfer prospects of one MAC address. Because the present occupancy state is reliant on the past occupancy status, the transfer possibility and transfer possibility matrix may be utilized to measure those type of methods. The transfer matrix may be stated as follows, assuming that an occupant's location in a place is either "in" or "out."

$$tpm\Big|_{yK} = \begin{bmatrix} y_K^{j-0} & y_K^{j-j} \\ y_K^{0-0} & y_K^{0-j} \end{bmatrix} \tag{34}$$

The transition probability matrix of one load is denoted as tpm_{yK}. In the transfer matrix, y_K^{j-0} and y_K^{j-j} indicate the noticed probability that single inhabitant whose position is "in" at the present period in any case be "out" and "in" at the following period, correspondingly, at the following period y_K^{0-0} and y_K^{0-j} signify the noticed possibility that one inhabitant whose position is "out" at the present period intermission would be "out" and "in" in the next period intermission. The possibility might be computed using Bayesian models and the observed conditional probability. For example

$$y_K^{j-j} = p\left(state\ observed = j \middle| state\ observed = j\right) \tag{35}$$

The one MAC address occupied probability is

$$y_K^{j-j} = \frac{\sum M_{1-1}}{\sum M_{1-1} + \sum M_{1-0}} \tag{36}$$

$$y_K^{0-0} = \frac{\sum M_{0-0}}{\sum M_{0-0} + \sum M_{0-1}} \tag{37}$$

where M_{1-1} is the recurrence in which the possession grade changed from "in" to "in" and M_{1-0} is the frequencies in which the possession grade changed from "in" to "out" respectively. Similarly, M_{0-0} and M_{0-1} address the frequencies in which the possession grade changed from "out" to "out" and from "out" to "in" individually. As the estimated frequency changes, the preventative education database will be automatically updated. The transfer probability will be adjusted at the next estimate as the training database is refreshed. Because each MAC address in the load is given a probability, each MAC address may be represented as follows:

$$y_K = \left\{ y_K^{mac}, y_K^{0-j}, y_K^{j-j} \right\} \tag{38}$$

Update the input vector in the following,

$$Y(T) = \left\{ y_1^{mac}, y_1^{0-j}, y_1^{j-j}, y_2^{mac}, y_2^{0-j}, y_2^{j-j}, \ldots y_K^{mac}, y_K^{0-j}, y_K^{j-j} \right\} \tag{39}$$

After that, the feature layer may be structured as follows:

$$f(T) = \{Y(T), Y(T-1), Y(T-2), \ldots . Y(T - \Delta T)\} \tag{40}$$

The length of time window is ΔT and at time T the vector of the Feature layer is *f*(*T*). Assuming the amount of MAC reports in the time window is K, then

$$f(T) = \left\{ y_1^{mac}, y_1^{0-j}, y_1^{j-j}, y_2^{mac}, y_2^{0-j}, y_2^{j-j}, \ldots y_K^{mac}, y_K^{0-j}, y_K^{j-j} \right\} \tag{41}$$

At regular intervals, the environment layer retains the hidden layer feedback signal, acting as a short-term memory to stress professional dependency. The rear cover layer's output may be structured as follows:

$$h(T) = g\left(\omega^1 D(T-1) + \omega^2 \left(f(T) \right) \right) \tag{42}$$

```
Input       : node list, task list and task history
Output      : cost function for CPU allocation
1   Initialize the values for the input parameters
2   Format for the MAC address is
    $Y(T) = \{y_1, y_2, \ldots, y_j, \ldots, y_l\}$
3   Compute the hidden layers neurons by
    $h(T) = Z_1^t * Y(T) + a$
4   Apply the transition probability matrix as
    $tpm\Big|_{yK} = \begin{bmatrix} y_K^{j-0} & y_K^{j-j} \\ y_K^{0-0} & y_K^{0-j} \end{bmatrix}$
5   Estimate the one MAC address occupied probability
    $y_K^{j-j} = \frac{\sum M_{1-1}}{\sum M_{1-1} + \sum M_{1-0}}$ and $y_K^{0-0} = \frac{\sum M_{0-0}}{\sum M_{0-0} + \sum M_{0-1}}$
6   Formatted each MAC address can be
    $y_K = \{y_K^{mac}, y_K^{0-j}, y_K^{j-j}\}$
7   Update the input vector in the following
    $Y(T) = \{y_1^{mac}, y_1^{0-j}, y_1^{j-j}, y_2^{mac}, y_2^{0-j}, y_2^{j-j}, \ldots y_K^{mac}, y_K^{0-j}, y_K^{j-j}\}$
8   Calculate output of the back cover layer
    $h(T) = g(\omega^1 D(T-1) + \omega^2 (f(T)))$
9   Evaluate the context layer output by
    $D(T-1) = \alpha D(T-2) + h(T-1)$
10  Update the cost function by
    $e = \sum_{T-1}^{M} [x(T) - c(T)]^2$
11  End
```

Algorithm 3 Pre-virtual CPU allocation using FARNN technique

The output of the context layer is

$$D(T-1) = \alpha D(T-2) + h(T-1) \tag{43}$$

where *h*(*T*) is referred as the output vector of the Hidden layer at time interval T, and D is the output vector of Context layer. ω^1 is stated as the joining mass from the Context layer to the Hidden layer, and ω^2 is the joining mass from the Feature layer to the Hidden layer. A is the self-connected comment gain factor. G (•) represents the Hidden layer's activation function. The mode of activation has been set to

$$g(y) = \frac{1}{1 + E^{-y}} \tag{44}$$

The following is an example of a signal change from the Hidden film to the Output film:

$$x(T) = \omega^3 h(T) = \omega^{3*} g\left(\omega^1 D(T-1) + \omega^2 f(T) \right) \tag{45}$$

where is the output variable at period T, which in this case is the expected possession. ω^3 is the joining mass from the Hidden layer to the Output layer. The following is the cost function for updating and learning connection weights:

$$e = \sum_{T-1}^{M} [x(T) - c(T)]^2 \tag{46}$$

c (t) is the actual occupancy output, and M is the size of training time samples. Algorithm 3 describes the process of pre-virtual CPU allocation.

Task load monitoring using DCNN method

There are five steps to the job load monitoring function: Data collecting and data filtering are the first two steps in the data collection process. 3) data gathering 4) examination of data 5) Issue a warning and file a complaint. Processing time, CPU speed from CPU probe, memory use, memory retrieval delay, power consumption, power consumption from power analysis, frequency, latency, and delay are all examples of information or quantity that the monitoring system should gather through various inquiries. Consider essential features of data gathering, such as structure, tactics, updating approaches, and kinds, to classify it. We employ a deep convolutional neural network (DCNN) to measure job load in this article. In DCNN, the scroll layer contains numerous filters that correspond to the intriguing local forms. The result is forwarded to a non-linear implementation function to generate a functional map. Also adjust the functional map that was constructed to reduce the calculated values by changing the properties. Stacking the scroll layers at the DCNN's front end separates the local attributes from the source data at first, and then gradually adds volume as the next abstract layer is provided. A well-trained layer produces a new representation of the original form that can be classified most successfully. For this purpose, the spiral layer is also called the functional sample layer. An assortment with several fully connected layers is attached at the end of the coil layer. For the training set samples,

$$n = \left\{ \left(y^{(j)}, x^{(j)} \right) \right\}, \ j = 1, 2, \ldots, n \tag{47}$$

Each sample has a feature vector $y^{(j)}$ and a label $x^{(j)}$ to go with it. By introducing the loss function, we may obtain the error. As demonstrated in following equation, the loss function has an overall error and a time order.

$$I(z, a) \approx \frac{1}{m} \sum_{j=1}^{m} k\left(H_{\{z,a\}} \left(y^{(j)}, x^{(j)} \right) \right) + \lambda \sum_{j,i} z_{j,i}^{2} \tag{48}$$

Here, z represents the weight and 'a' denotes the bias value respectively. Also, the size of the batch is represented as m. The hyper parameter λ error regulates and controls error values. The dissimilarity amongst the created assessment and the real assessment is measured in square metres. It's worded like this:

$$D = \frac{1}{2M} \sum_{y} \left\| x(y) - b(y) \right\|^{2} \tag{49}$$

When calculating two gradients, the coefficient 1/2 is a normalization group that cancels the coefficient. Further derivatives can be simplified without causing side effects as a result of this. Also can modify the weight and offset to reduce losses depending on the look of the slope.

$$\Delta\omega = \left(b(y) - x(y) \right) \sigma^{'}(w) y \tag{50}$$

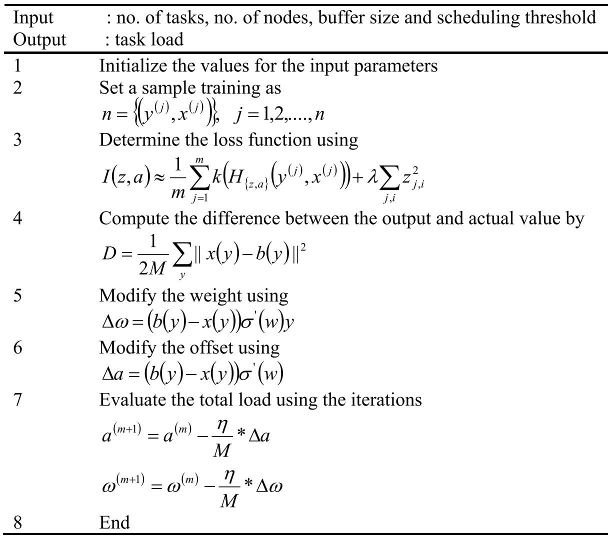
$$\Delta a = \left(b(y) - x(y) \right) \sigma^{'}(w) \tag{51}$$

In the neuron, the input is denoted as w; the activation function is represented as σ; the change in the weight is referred as $\Delta\omega$ and the variation of the offset is stated as Δa respectively.

$$\omega^{(m+1)} = \omega^{(m)} - \frac{\eta}{M}^{*} \Delta\omega \tag{52}$$

$$a^{(m+1)} = a^{(m)} - \frac{\eta}{M}^{*} \Delta a \tag{53}$$

The learning rate is represented as η; the m^th iteration weight and offset are denoted as $\omega^{(m)}$ and $a^{(m)}$ respectively. The total number of loads is represented as M respectively. In Algorithm 4, we describe the working function of the task load monitoring using DCNN method.



```
Input   : no. of tasks, no. of nodes, buffer size and scheduling threshold
Output  : task load
Initialize the values for the input parameters
Set a sample training as
    $n = \{(y^{(j)}, x^{(j)})\}, \ j = 1,2,....,n$
Determine the loss function using
    $I(z,a) \approx \frac{1}{m}\sum_{j=1}^{m} k(H_{\{z,a\}}(y^{(j)}, x^{(j)})) + \lambda\sum_{j,i} z_{j,i}^{2}$
Compute the difference between the output and actual value by
    $D = \frac{1}{2M}\sum_{y} \| x(y) - b(y) \|^{2}$
Modify the weight using
    $\Delta\omega = (b(y) - x(y))\sigma^{'}(w)y$
Modify the offset using
    $\Delta a = (b(y) - x(y))\sigma^{'}(w)$
Evaluate the total load using the iterations
    $a^{(m+1)} = a^{(m)} - \frac{\eta}{M} * \Delta a$
    $\omega^{(m+1)} = \omega^{(m)} - \frac{\eta}{M} * \Delta\omega$
End
```

Algorithm 4 Task load monitoring using DCNN method

Simulation results and analysis

In this part, we develop experimentations to test and assess the proposed dynamic scalable task scheduling (DSTS) model, and the simulation results are associated to current state-of-the-art models including ADATSA, LAEAS, PSOS, and the K8S planning machine.

- To overcome the repeating scheduling issue, a self-accommodating task planning algorithm (ADATSA) is used [33]. The approach reduces the reliance of existing vibrant planning strategies on container cloud architecture and improves the connection between jobs and their runtime environments.
- In the cloud system, the Learning automata based energy-aware scheduling (LAEAS) algorithm [37] is employed for real-time job planning.
- In a container cloud context, the performance-based service oriented scheduling (PSOS) [38] has been utilised to handle planning problems such as average latency of service instances, resource consumption, and balancing.

- Unlike Borg and Omega, which were built as completely Google-internal systems, the Kubernetes (K8S) scheduling engine [39] is open source.

Dataset description

Kubernetes (v1.16.2) was used to create an experimental setup on 53 servers with the similar specs as the investigational stage, comprising 3, 50 master and slave nodes. Furthermore, we utilised Python 3.7 as the major programming language for quality analysis implementation, with Anaconda Navigator integration and spyder and Jupyter as execution environments. The number of tasks in this simulation has been separated into five categories: task 1, task 2, task 3, task 4, and task 5. In job 1, we may use static scheduling with 128core and 64core CPU oriented resources as master and slave, respectively. In task 2, we may use memory-oriented resources master and slave of 256GB and 128GB, respectively, to create dynamic scheduling. In task 3, we may use time-based static scheduling with 1000GB master and slave disc oriented resources, respectively. Task 4 allows us to configure time-based dynamic scheduling with bandwidth-oriented master and slave resources of 10Gbps and 10Gbps, respectively. With the resource non-oriented master and slave as 3 and 50, we may examine test quality in job 5. Where resource non-oriented apps are ones in which the application's resource needs are composed and there is no partiality for resources. Table 2 summarises the job partitioning and resource requirements. We employed recurrent distributions to mimic large-scale uses distribution due to a shortage of apps. The experiment began with a total of 100 applications, including 20 for each category of application. Table 3 describes the super-parameter settings of proposed optimization algorithm.

Table 2 Dataset descriptions

Tasks	Scheduling	Resources	Node resources	
			Master	Slave
1	Static	CPU oriented (core)	128	64
2	Dynamic	Memory oriented (GB)	256	128
3	Static-time	Disk oriented (GB)	1000	1000
4	Dynamic-time	Bandwidth oriented (Gbps)	10	10
5	QoS evaluation	Resource non-oriented	3	50

Table 3 Optimization algorithm super-parameter settings

Parameters	Value
Population size	80
Crossover probability	0.8
Mutation probability	0.2
Maximum number of generation	200
Swarm size	80
Maximum number of iteration	200

Performance evaluation metrics

In this section, the simulation results of proposed DSTS classic is associated with the existing state-of-art models such as ADATSA, LAEAS, PSOS and K8S planning engine in terms of different service quality evaluation metrics are resource imbalance degree (D_{Id}), resource residual degree (D_{Rd}), response time (R_T) and throughput (T_H). The particulars of appropriate metrics are defined as proceeds:

$$D_{Id} = \sum_{i=1}^{N} \frac{L_r(\alpha_i)}{N} \tag{54}$$

$$D_{Rd} = \sum_{i=1}^{N} \frac{S_r(\beta_i)}{N} \tag{55}$$

$$R_T = \frac{1}{N_{app}} \sum_{j=1}^{N_{app}} R_T \ WS_{app} \tag{56}$$

$$T_H = \frac{N_{req} \ WS_{app}}{T_{end} \ WS_{app} - T_{start} \ WS_{app}} \tag{57}$$

where $L_r(\alpha_i)$ and $S_r(\beta_i)$ represents node resource imbalance degree (ref. eqn [18].) and node resource residual degree (ref. eqn [19].) respectively for N number of node resources. The response delay of web application represents as WS_{app} and T_{end}, T_{start} denotes the start and end time of the test respectively.

Comparative analysis

Result comparison of Task-1

The influence of tasks on static scheduling performance of our new DSTS model is compared to that of the current ADATSA, LAEAS, PSOS, and K8S models in this scenario. The proposed and current task scheduling models are compared in terms of resource imbalance degree (DId) in Fig. 2. We can see from this graph that the DSTS model of static scheduling outperforms the ADATSA, LAEAS, PSOS, and K8S models. The suggested DSTS model has a resource imbalance degree (DId) of 12.698%, 10.000%, 7.895%, and 6.173%, respectively, lower than the current ADATSA, LAEAS, PSOS, and K8S models. Figure 3 shows the comparative analysis

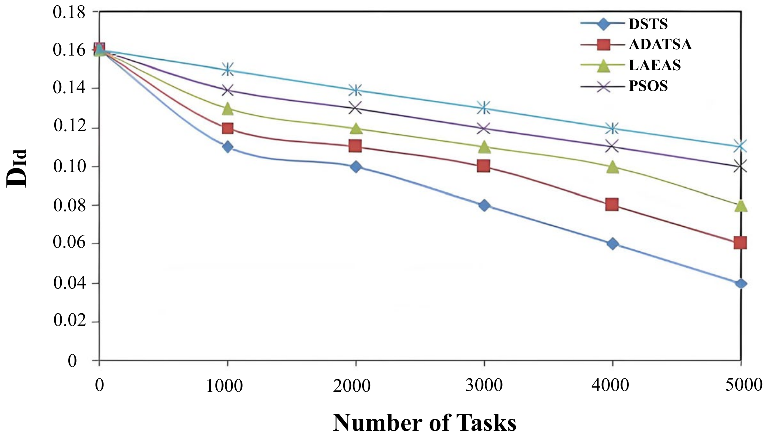


Fig. 2 Comparative analysis of resource imbalance degree (D_{Id}) (Task-1)

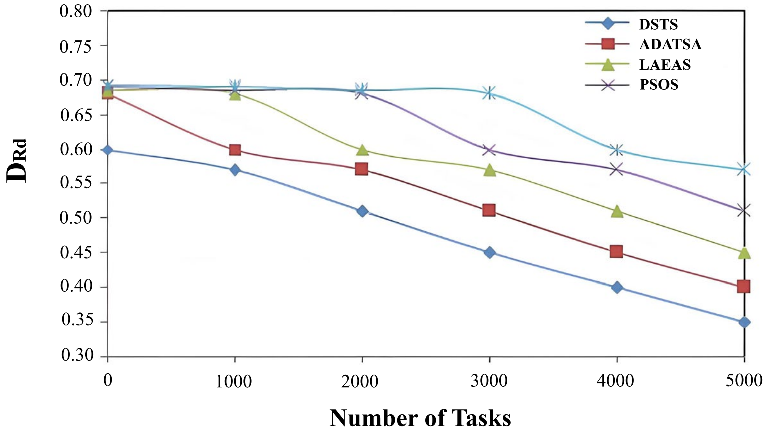


Fig. 3 Comparative analysis of resource residual degree (D_{Rd}) (Task-1)

of resource residual degree (D_{Rd}) for the proposed and existing task scheduling models. We can see from this graph that the DSTS model of static scheduling outperforms the ADATSA, LAEAS, PSOS, and K8S models. The resource residual degree (D_{Rd}) of proposed DSTS model is 10.280%, 8.155%, 6.426% and 4.695% lower than the existing ADATSA, LAEAS, PSOS and K8S models respectively.

Result comparison of Task-2

The influence of tasks on the dynamic scheduling presentation of our suggested DSTS model is associated to that of the current ADATSA, LAEAS, PSOS, and K8S models in this scenario. Figure 4 shows the comparative analysis of resource imbalance degree (D_{Id}) for the proposed and existing task scheduling models. We can see from this graph that the DSTS dynamic scheduling model outperforms the ADATSA, LAEAS, PSOS, and K8S models. The resource imbalance degree (D_{Id}) of proposed DSTS model is 15.275%, 9.285%, 8.590% and 6.699% lower than the existing ADATSA, LAEAS, PSOS and K8S models respectively. Figure 5 shows the comparative analysis of resource residual degree (D_{Rd}) for the proposed and existing task scheduling models. We can see from this graph that the DSTS model of dynamic scheduling outperforms the ADATSA, LAEAS, PSOS, and K8S models. The resource residual degree (D_{Rd}) of proposed DSTS model is 11.710%, 8.555%, 6.740% and 5.462% lower than the existing ADATSA, LAEAS, PSOS and K8S models respectively.

Result comparison of Task-3

In this scenario, the influence of tasks on our proposed DSTS model's time-based static scheduling performance is compared to the current ADATSA, LAEAS, PSOS, and K8S models. Figure 6 shows the comparative analysis of

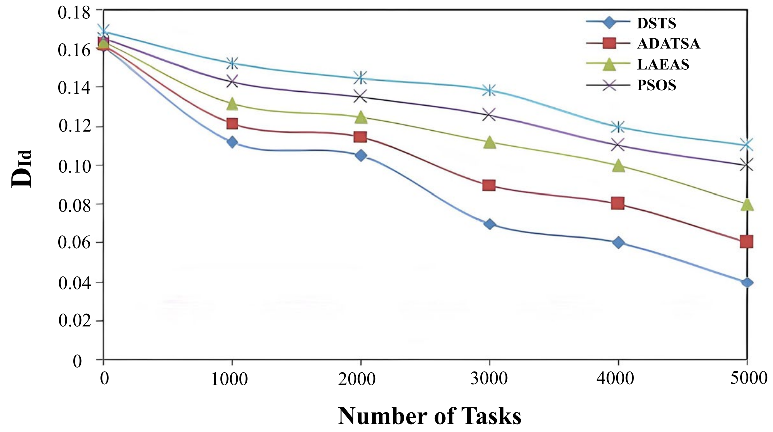


Fig. 4 Comparative analysis of resource imbalance degree (D_{Id}) (Task-2)

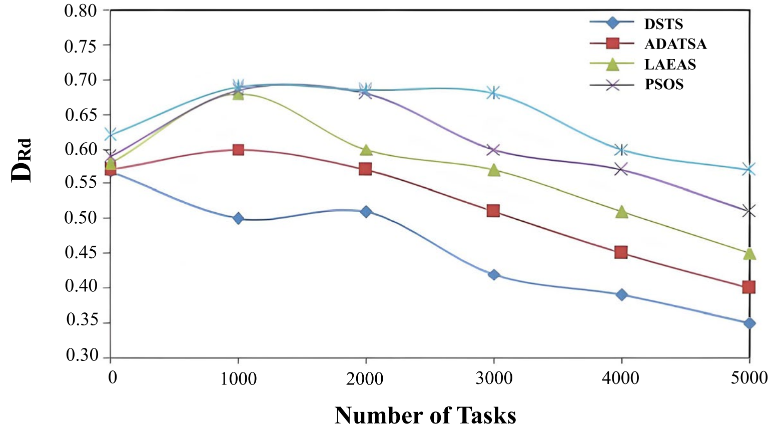


Fig. 5 Comparative analysis of resource residual degree (D_{Rd}) (Task-2)

resource imbalance degree (D_{Id}) with respect to time for the proposed and existing task scheduling models. We can see from this graph that the DSTS model of static scheduling outperforms the ADATSA, LAEAS, PSOS, and K8S models. The resource imbalance degree (D_{Id}) of proposed DSTS model is 15.146%, 15.275%, 9.285% and 8.590% lower than the existing ADATSA, LAEAS, PSOS and K8S models respectively. Figure 7 shows the comparative analysis of resource residual degree (D_{Rd}) with respect to time for the proposed and existing task scheduling models. We can see from this graph that the DSTS model of static scheduling outperforms the ADATSA, LAEAS, PSOS, and K8S models in terms of performance. The resource residual degree (D_{Rd}) of proposed DSTS model is 6.796%, 11.710%, 8.555% and 6.740% lower than the existing ADATSA, LAEAS, PSOS and K8S models respectively.

Result comparison of Task-4

In this scenario, the influence of tasks on our proposed DSTS model's time-based dynamic scheduling performance is compared to the current ADATSA, LAEAS, PSOS, and K8S models. Figure 8 shows the comparative analysis of resource imbalance degree (D_{Id}) with respect to time for the proposed and existing task scheduling models. We can see from this graph that the DSTS model of static scheduling outperforms the ADATSA, LAEAS, PSOS, and K8S models. The resource imbalance degree (D_{Id}) of proposed DSTS model is 13.763%, 15.146%, 12.878% and 11.781% lower than the existing ADATSA, LAEAS, PSOS and K8S models respectively. Figure 9 shows the comparative analysis of resource residual degree (D_{Rd}) with respect to time for the proposed and existing task scheduling models. We can see from this graph that the DSTS model of static scheduling outperforms the ADATSA, LAEAS, PSOS, and K8S models.

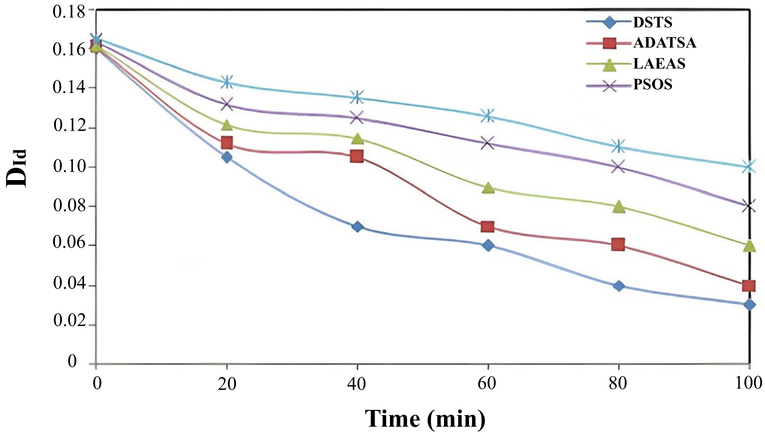


Fig. 6 Comparative analysis of resource imbalance degree (D_{Id}) with time (Task-3)

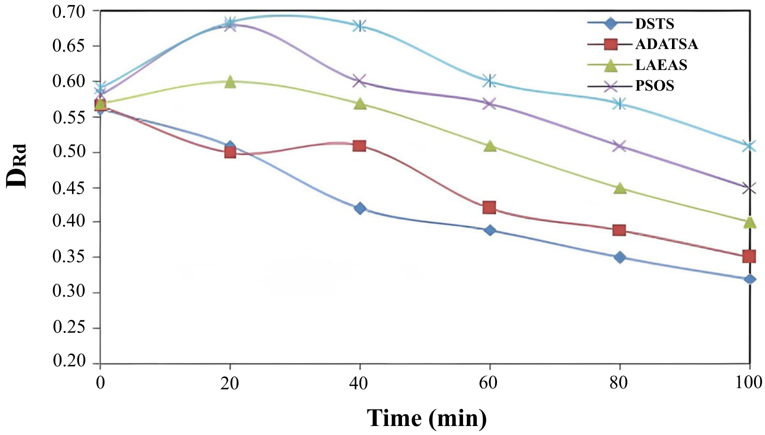


Fig. 7 Comparative analysis of resource residual degree (D_{Rd}) with time (Task-3)

The resource residual degree (D_{Rd}) of proposed DSTS model is 6.703%, 6.796%, 11.710% and 8.555% lower than the existing ADATSA, LAEAS, PSOS and K8S models respectively.

Result comparison of Task-5

In this scenario, the effect of our proposed DSTS model's quality validation is compared to the current ADATSA, LAEAS, PSOS, and K8S models. Figure 10 shows the comparative analysis of resource imbalance degree (D_{Id}) with respect to time for the proposed and existing task scheduling models. We can see from this graph that the DSTS model of static scheduling outperforms the ADATSA, LAEAS, PSOS, and K8S models. The resource imbalance degree (D_{Id}) of proposed DSTS model is 13.965%, 13.763%, 15.146% and 12.878% lower than the existing ADATSA, LAEAS, PSOS and K8S models respectively. Figure 11 shows the comparative analysis of resource residual degree (D_{Rd}) with respect to time for the proposed and existing task scheduling models. We can see from this graph that the DSTS model of static scheduling outperforms the ADATSA, LAEAS, PSOS, and K8S models in terms of performance. The resource residual degree (D_{Rd}) of proposed DSTS model is 13.445%, 6.703%, 6.796% and 11.710% lower than the existing ADATSA, LAEAS, PSOS and K8S models respectively.

Table 4 describes the performance comparison of proposed and existing task scheduling in terms of response time (R_T) and throughput (T_H) with varying simulation time. The average response time (R_T) of proposed DSTS model is 25.448%, 32.616%, 37.814% and 40.502% higher than the existing ADATSA, LAEAS, PSOS and K8S models respectively. Figure 12 gives the graphical representation of proposed and existing task scheduling models. The average throughput (T_H) of proposed DSTS model is

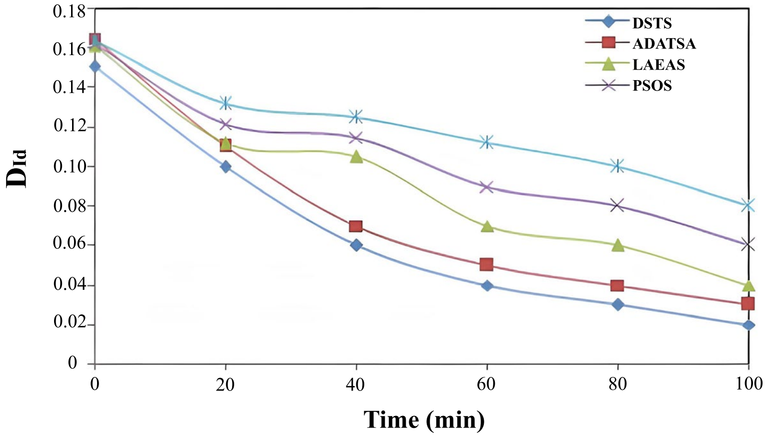


Fig. 8 Comparative analysis of resource imbalance degree (D_{Id}) with time (Task-4)

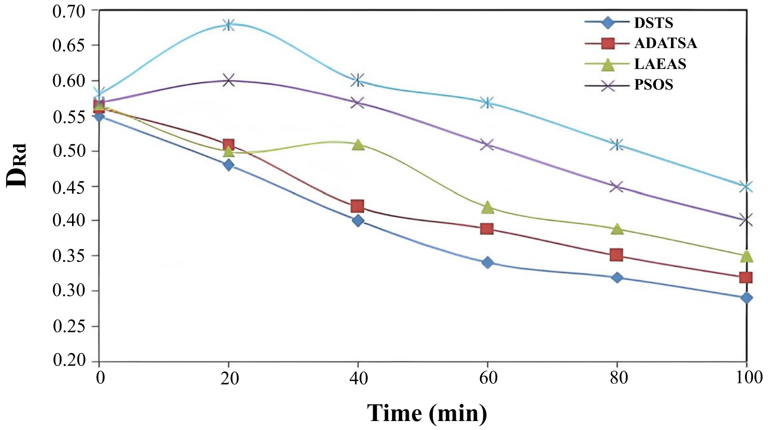


Fig. 9 Comparative analysis of resource residual degree (D_{Rd}) with time (Task-4)

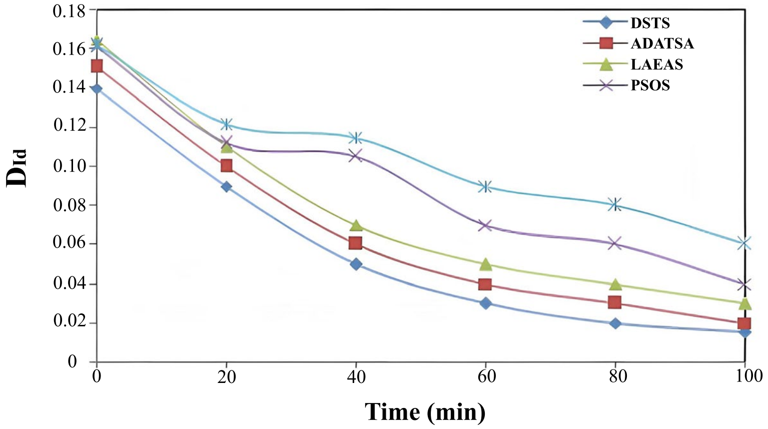


Fig. 10 Comparative analysis of resource imbalance degree (D_{Id}) with time (Task-5)

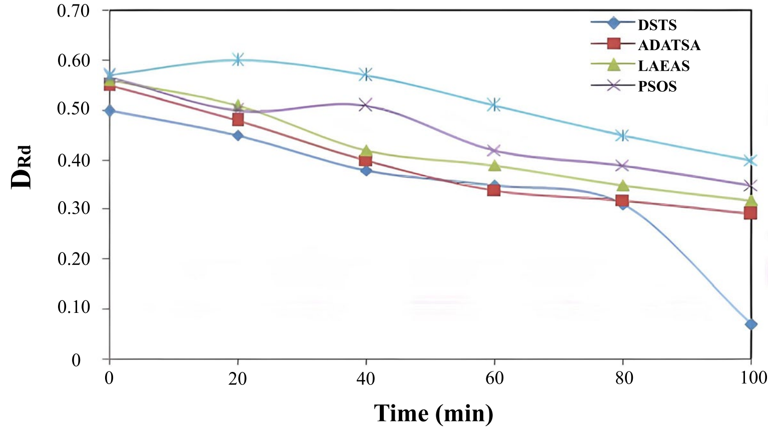


Fig. 11 Comparative analysis of resource residual degree (D_{Rd}) with time (Task-5)

Table 4 Comparative analysis of quality of service metrics

Models	Response time (R_T) (ms)					Throughput (T_H)				
	20	40	60	80	100	20	40	60	80	100
DSTS	600	580	560	550	500	150	180	210	220	250
ADATSA	500	480	400	380	320	120	130	135	140	150
LAEAS	480	400	380	320	300	100	120	130	135	140
PSOS	450	380	320	300	285	80	100	120	130	135
K8S	430	360	310	290	270	75	90	100	120	130

33.168%, 38.119%, 44.059% and 49.010% higher than the existing ADATSA, LAEAS, PSOS and K8S models respectively. Figure 13 gives graphical representation of proposed and existing task scheduling models. Figure 14 denotes the runtime overhead of the proposed and existing task scheduling models. The plot clearly depicts average runtime overhead of the proposed DSTS model is 12.356%, 15.09%, 18.367% and 21.578% lower than the existing ADATSA, LAEAS, PSOS and K8S models respectively.

Case study

In the past, Kaplan used the Amazon Elastic Compute cloud to host its applications. Working engineers were required to manually update applications, and on average there were four dedicated Amazon EC2 hosts. Rowan Drabo, head of Kaplan cloud operations, said the application update would take hours to take effect. Cost analysis shows that we spend more than $ 500 per month on the Amazon Elastic Compute cloud. After switching to micro-service-based architecture with Amazon's flexible container service and containers, Kaplan saved significant costs. "We currently have more than 500 containers in production," Drabo said. We have reduced the number of Amazon Flexible Compute cloud events by 70%, resulting in 40% cost savings per application. Using our proposed Dynamic Scalable Task Scheduler (DSTS) for automated container delivery, Kaplan allows you to reduce deployment time, increase the frequency of updates and improve developer satisfaction.

Conclusion

For dynamic scalable task scheduling (DSTS) in a container cloud context, we suggested a hybrid optimum and deep learning approach. The succeeding are the major influences made in this paper:

1. A modified multi-swarm coyote optimization (MMCO) method for scaling virtual resources in containers to improve customer service level agreements.
2. A modified pigeon-inspired optimization (MPIO) algorithm is for task clustering and fast adaptive

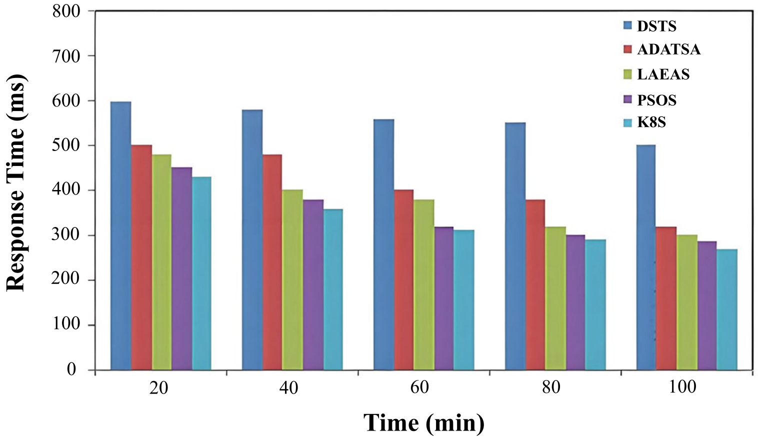


Fig. 12 Comparative analysis of response time (R_T) (Task-5)

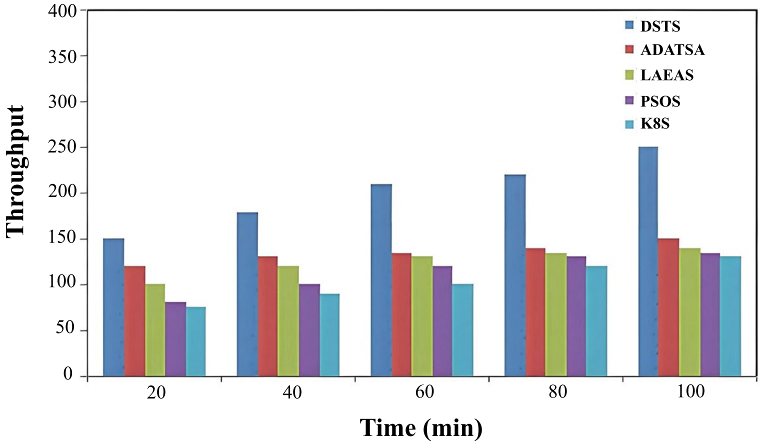


Fig. 13 Comparative analysis of Throughput (T_H) (Task-5)

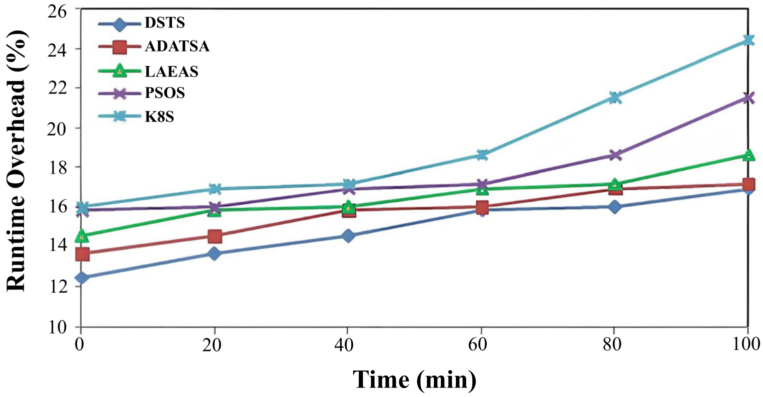


Fig. 14 Comparative analysis of runtime overhead

feedback recurrent neural network (FARNN) for pre-virtual CPU allocation to ensure priority based scheduling.

3. Task load monitoring mechanism is designed based on deep convolutional neural network (DCNN) which achieves dynamic scheduling based on priority.

After the recreation outcomes, we concluded that the simulation results of projected DSTS model is very effective compared to the existing task scheduling models in terms of excellence of service metrics are resource imbalance degree (D_{Id}), resource residual degree (D_{Rd}), response time (R_T) and throughput (T_H). In future, we extend our DSTS model which combine with the optimization algorithm to optimize joint problems i.e. resource allocation and task scheduling in container cloud environment.

Authors' contributions

Mr. Saravanan Muniswamy has made substantial contributions to design and in drafting the manuscript. Mr. Vignesh Radhakrishnan has made HIS contributions in acquisition of data and interpretation of data. The author(s) read and approved the final manuscript.

Funding

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Declarations

Competing interests

The authors have no relevant financial or non-financial interests to disclose.

Received: 8 March 2022 Accepted: 26 July 2022
Published online: 30 August 2022

References

1. Wang B, Qi Z, Ma R, Guan H, Vasilakos AV (2015) A survey on data center networking for cloud computing. Comput Netw 91:528–547
2. González-Martínez JA, Bote-Lorenzo ML, Gómez-Sánchez E, Cano-Parra R (2015) Cloud computing and education: a state-of-the-art survey. Comput Educ 80:132–151
3. Khan AN, Kiah MM, Khan SU, Madani SA (2013) Towards secure mobile cloud computing: a survey. Futur Gener Comput Syst 29(5):1278–1299
4. Xie XM, Zhao YX (2013) Analysis on the risk of personal cloud computing based on the cloud industry chain. J China Univ Posts Telecommun 20:105–112
5. Han Y, Luo X (2013) Hierarchical scheduling mechanisms for multilingual information resources in cloud computing. AASRI Proc 5:268–273
6. Bose R, Luo XR, Liu Y (2013) The roles of security and trust: comparing cloud computing and banking. Procedia Soc Behav Sci 73:30–34
7. Elamir AM, Jailani N, Bakar MA (2013) Framework and architecture for programming education environment as a cloud computing service. Proc Technol 11:1299–1308
8. Tsertou A, Amditis A, Latsa E, Kanellopoulos I, Kotras M (2016) Dynamic and synchromodal container consolidation: the cloud computing enabler. Transp Res Proc 14:2805–2813
9. Kong W, Lei Y, Ma J (2016) Virtual machine resource scheduling algorithm for cloud computing based on auction mechanism. Optik 127(12):5099–5104
10. Moschakis IA, Karatza HD (2015) A meta-heuristic optimization approach to the scheduling of bag-of-tasks applications on heterogeneous clouds with multi-level arrivals and critical jobs. Simul Model Pract Theory 57:1–25
11. Singh S, Chana I (2015) QRSF: QoS-aware resource scheduling framework in cloud computing. J Supercomput 71(1):241–292
12. Lin J, Zha L, Xu Z (2013) Consolidated cluster systems for data centers in the cloud age: a survey and analysis. Front Comput Sci 7(1):1–19
13. Kertész A, Dombi JD, Benyi A (2016) A pliant-based virtual machine scheduling solution to improve the energy efficiency of iaas clouds. J Grid Comput 14(1):41–53
14. Musa IK, Walker SD, Owen AM, Harrison AP (2014) Self-service infrastructure container for data intensive application. J Cloud Comput 3(1):1–21
15. Choe R, Cho H, Park T, Ryu KR (2012) Queue-based local scheduling and global coordination for real-time operation control in a container terminal. J Intell Manuf 23(6):2179–2192
16. Nam H, Lee T (2013) A scheduling problem for a novel container transport system: a case of mobile harbor operation schedule. Flex Serv Manuf J 25(4):576–608
17. Bian Z, Li N, Li XJ, Jin ZH (2014) Operations scheduling for rail mounted gantry cranes in a container terminal yard. J Shanghai Jiaotong Univ Sci 19(3):337–345
18. Zhang R, Yun WY, Kopfer H (2010) Heuristic-based truck scheduling for inland container transportation. OR Spectr 32(3):787–808
19. Briskorn D, Fliedner M (2012) Packing chained items in aligned bins with applications to container transshipment and project scheduling. Mathem Methods Oper Res 75(3):305–326
20. Briskorn D, Angeloudis P (2016) Scheduling co-operating stacking cranes with predetermined container sequences. Discret Appl Math 201:70–85
21. Zhao D, Mohamed M, Ludwig H (2018) Locality-aware scheduling for containers in cloud computing. IEEE Trans Cloud Comput 8(2):635–646
22. Liu B, Li P, Lin W, Shu N, Li Y, Chang V (2018) A new container scheduling algorithm based on multi-objective optimization. Soft Comput 22(23):7741–7752
23. Lin M, Xi J, Bai W, Wu J (2019) Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. IEEE Access 7:83088–83100
24. Adhikari M, Srirama SN (2019) Multi-objective accelerated particle swarm optimization with a container-based scheduling for Internet-of-Things in cloud environment. J Netw Comput Appl 137:35–61
25. Ranjan R, Thakur IS, Aujla GS, Kumar N, Zomaya AY (2020) Energy-efficient workflow scheduling using container-based virtualization in software-defined data centers. IEEE Trans Industr Inform 16(12):7646–7657
26. Chen Q, Oh J, Kim S, Kim Y (2020) Design of an adaptive GPU sharing and scheduling scheme in container-based cluster. Clust Comput 23(3):2179–2191
27. Hu Y, Zhou H, de Laat C, Zhao Z (2020) Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. Futur Gener Comput Syst 102:562–573
28. Rajasekar P, Palanichamy Y (2020) Scheduling multiple scientific workflows using containers on IaaS cloud. 7621–7636 (2021) J Ambient Intell Humaniz Comput 1–16
29. Menouer T (2021) KCSS: Kubernetes container scheduling strategy. J Supercomput 77(5):4267–4293
30. Li C, Zhang Y, Luo Y (2021) Neighborhood search-based job scheduling for IoT big data real-time processing in distributed edge-cloud computing environment. J Supercomput 77:1853–1878
31. Ahmad I, AlFailakawi MG, AlMutawa A, Alsalman L (2021) Container scheduling techniques: a survey and assessment. Journal of King Saud University-Computer and Information Sciences 34(2022):3934-3947
32. Rausch T, Rashed A, Dustdar S (2021) Optimized container scheduling for data-intensive serverless edge computing. Futur Gener Comput Syst 114:259–271
33. Zhu L, Huang K, Hu Y, Tai X (2021) A self-adapting task scheduling algorithm for container cloud using learning automata. IEEE Access 9:81236–81252
34. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I et al (2010) A view of cloud computing. Commun ACM 53(4):50–58
35. Gawali MB, Shinde SK (2018) Task scheduling and resource allocation in cloud computing using a heuristic approach. J Cloud Comp 7:4

36. Gawali MB, Gawali SS (2021) Optimized skill knowledge transfer model using hybrid Chicken Swarm plus Deer Hunting Optimization for human to robot interaction. Knowl-Based Syst 220:106945
37. Sahoo S, Sahoo B, Turuk AK (2018) An energy-efficient scheduling framework for cloud using learning automata. In: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE, Bangalore, India. pp 1–5
38. Li H, Wang X, Gao S, Tong N (2020) A service performance aware scheduling approach in containerized cloud. In: 2020 IEEE 3rd International Conference on Computer and Communication Engineering Technology (CCET). IEEE, Beijing, China. pp 194–198
39. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J (2016) Borg, omega, and kubernetes. Commun ACM 59(5):50–57

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.