Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Task grouping and optimized deep learning based VM sizing for hosting containers as a service

Manoj Kumar Patra[1], Bibhudatta Sahoo[1], Ashok Kumar Turuk[1] and Sanjay Misra[2,3*]

Abstract

Containers as a service (CaaS) are a kind of services that permits the organization to handle the containers more effectively. Containers are lightweight, require less computing resources, portable, and facilitate better support for microservices. In the CaaS model, containers are deployed in virtual machines, and the virtual machine runs on the physical machine. The objective of this paper is to estimate the resource required by a VM to execute a number of containers. VM sizing is a directorial process where the system administrators have to optimize the allocated resources within the permitted virtualized space. In this work, the VM sizing is carried out using the Deep Convolutional Long Short Term Memory Network (Deep-ConvLSTM), where the weights are updated by Fractional Pelican Optimization (FPO) Algorithm. Here, the FPO is configured by hybridizing the concept of Fractional Calculus (FC) within the updated location of the Pelican Optimization Algorithm (POA). Moreover, the task grouping is done with Deep Embedded Clustering (DEC), where the grouping is established with respect to the various task parameters, such as task length, submission rate, scheduling class, priority, resource usage, task latency, and Task Rejection Rate (TRR). In addition, the performance analysis of VM sizing is done by taking 100, 200, 300, and 400 tasks. We got the best resource utilization of 0.104 with 300 tasks, a response time of 262ms with 100 tasks, and a TRR of 0.156 with 100 tasks and makespan of 0.5788 with 100 tasks.

Keywords Cloud computing, Container as a service, Long short term memory, Pelican optimization algorithm, Fractional calculus, Deep embedded clustering, Convolutional LSTM network

Introduction

With the progressive development in Information Technology (IT), a new substitute has emerged in the form of cloud computing for the conventional computing methods to provide services to clients without any constraint on location or time, thereby enabling access to a pool of distributed computing resources to the users [1, 2]. Cloud computing provides various benefits in terms of reducing operational expenses and funds. It develops an environment based on networks that allow the distribution of resources and computations irrespective of the user's location [3]. Recently, several firms are migrating to cloud computing applications due to the high efficiency, Quality of Service (QoS), accessibility, and reduced cost. Moreover, the pool of resources offers high scalability and flexibility because of the dispersed nature of storage and computational resources [4].

Services in the cloud are provided based on three service models such Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service(SaaS). In the IaaS service model, web portals and the Application Program Interface (API) provide the hardware

*Correspondence:
Sanjay Misra
ssopam@gmail.com; sanjay.misra@hiof.no
[1] Department of Computer Science and Engineering, National Institute of Technology, Rourkela 769 008, India
[2] Department of Computer Science and Communication, Østfold University College, Halden, Norway
[3] Present Address: Department of Applied Data Science, Institute of Energy Technology, Halden, Norway

necessary for storage and computation. The PaaS is a platform-oriented model where application-oriented services, like database services and scripting environments, are provided, and the SaaS offers app-based services [5]. In addition to the three classical models mentioned above, a new model known as Container as a Service (CaaS) has been devised, making cloud computing more powerful and efficient in terms of resource utilization. Virtualization using container have gained high acceptance due to the lightweight solution offered, which permits the bunching of data and applications in a simple and performance-directed way that makes them suitable to be executed on various cloud frameworks [6]. A CaaS model comprises containers that are packed with applications. The Virtual Machines (VMs) run the containers, and each VM is hosted in Physical Machines (PMs) [7]. CaaS model lies in between the IaaS and PaaS [8]. Container refers to the virtualization technology operated at the system level, which can be positioned on PMs or VMs. The prime purpose of using CaaS is to provide an isolated atmosphere for the application execution. The container can share Operating System (OS) kernel compared to a VM, which requires all the resources available in the OS. Hence, it is a lightweight model requiring minimum time and resources [8, 9].

The inefficient utilization of resources can occur when more resources are allocated than necessary. These results in a workload executed by many PMs than that required [10]. The consumption of energy in CaaS can be reduced by considering the Dynamic Voltage and Frequency Scaling (DVFS), VM consolidation, or both. But the efforts would be meaningless without considering the customization of the VM sizes to enhance support of the deployed containers [8]. VM sizing can be defined as the measure of the total quantity of resources that have to be assigned to a VM. VM sizing is done with the objective of ensuring that the VM capacity is proportionate to the workload assigned to a VM [11]. The number of resources required by a VM wills be estimated by the Cloud Service Provider (CSP), which collects and analyses data associated with the VM. The CSP considers the applications that are executed in a VM as a black box [10]. Recently, Deep learning (DL) techniques using Recursive Neural Networks (RNN), Convolutional Neural Networks (CNN), etc., have a high ability to detect hidden information from the training data, and this can be utilized to analyze the data using black box approach [12].

The major aim of this research is the modeling of a VM sizing scheme with task grouping. Here, the VM sizing is done using Deep-ConvLSTM, where its weight is trained with an invented FPO algorithm. Moreover, the FPO algorithm is modeled by applying the FC concept in the updated POA location. Furthermore, the task grouping is carried out using DEC, which is done based on the task parameters, such as task length, submission rate, scheduling class, priority, resource usage, task latency, and task rejection rate. In the VM sizing, horizontal and vertical sizing are applied to each task group. The significant contributions of this research are:

- *FPO assisted Deep-ConvLSTM for VM sizing:* The VM sizing is carried out using Deep-ConvLSTM, which is trained with the invented FPO algorithm. Here, the devised FPO algorithm is modeled by including the concept of FC with the updated condition of POA such that the optimal location is updated. Thereby better prediction performance is attained.
- The proposed approach is validated with Google's cluster trace data set, which indicates that it outperforms the other techniques, such as VM size selection technique, Hurst exponent +Markov transition based VM scaling, Inter-Cloud Load Balancer (ICLB), VM size IaaS multi-tenant public cloud [6], Online Multi-Resource Feed-forward Neural Network (OM-FNN), Deep-ConvLSTM_Gradient Descendant (GD), and Deep-ConvLSTM ADAM for virtual machine sizing in CaaS cloud.

The rest of the paper is structured as follows; Section "Literature survey and motivation" portrays the literature survey of VM sizing techniques in the cloud, the cloud system model is explained in Section "System model", Section "Proposed task grouping and optimized deep learning based VM sizing for hosting CaaS" indicates the proposed methodology for hosting CaaS, and Section "Results and discussion" portrays the results and discussion of devised model. Section "Discussion" presents the numerical results and finally in Section "Conclusion", we conclude the paper with some future directions.

Literature survey and motivation

Literature survey

The challenges of conventional VM sizing techniques in the cloud are explained below. In [8], Piraghaj, S.F. et al., modeled the VM size selection technique for performing the VM sizing. The method achieved minimal energy utilization by decreasing the wastage of resources. However, it failed to reflect on predicting workload for estimating the actual container utilization. Lu, C.T. et al., [13] introduced the Hurst exponent and Markov transition-based VM scaling based on empirical cloud information to reflect the predicting overload. Moreover, this approach successfully enhanced the host availability by increasing the usage of inactive resources. Though, it was unsuccessful in aggregating more data to enhance the strengthening

of the transition matrix to improve accuracy. To progress the accuracy, Sotiriadis S. et al. [14] modeled the Inter-Cloud Load Balancer (ICLB) for configuring the VM. Here, the devised scheme successfully eradicates the downtime without affecting the flow of information. However, it failed to tune the method for defining thresholds based on real-time data utilization. Guo, Y. et al. [15] modeled the Shadow algorithm for performing the VM auto-scaling for devising the VM scaling scheme based on real-time data utilization. This method attained high scalability, which enabled the handling of a larger cloud, but it did not find any ways to scale the resources to minimize the count of tear-downs and VM boots. To diminish the tear-down and VM boots count, Alsadie D., et al. [16] designed the Dynamic Resource Allocation to minimize the energy usage in cloud computing. This method successfully minimized the energy utilization in cloud data centers. However, it produced a high overhead with the increased number of VMs. To diminish the computational overhead, Derdus, K.M. et al., [17] introduced the VM right-sizing IaaS multitenant public cloud for sizing the VM in cloud data centers. This method efficiently reduced the operating costs required by the CSP. However, it failed to use a VM allocation algorithm for estimating the peak resources needed before allocating them. To estimate the peak resources, Kenga, D.M. et al., [10] designed the VM sizing in IaaS multitenant public cloud for performing the VM sizing and predicting the resource usage. This method effectively predicted the resource utilization for multitenant IaaS cloud, but it was unsuccessful in seeing the deep learning approaches and various other cloud infrastructures to improve the performance. For attaining a superior outcome based on a deep learning scheme, Saxena, D. and Singh, A.K., [18] invented the Online Multi-Resource Feed-forward Neural Network (OM-FNN) for performing the autoscaling and VM allocation in the cloud data center. It instantaneously estimated numerous resource usage in reduced space and time complexity. Though, the developed approach failed to minimize the percentage of power saved with the rise in data center size.

Sareh Fotuhi Piraghaj, in her research [19], emphasizes different resource management strategies in the CaaS cloud system. She has used a clustering-based approach for task grouping and a virtual machine sizing technique for efficient resource utilization in CaaS. Jialei Liu et al. [20] explore the cloud resource utilization using container consolidation. The proposed approach considers usage prediction to achieve the objective, such as reducing the number of running virtual machines, migrating containers, and reducing energy consumption while satisfying SLA. Vasiliki Liagkou et al. [21] presented a CaaS model where container clustering is done and deployed in a virtual machine. They proposed a hedonic model for pricing that analyzes the provider's price and the correlation between different requirements. Weiwen Zhang et al. [22] try to improve resource utilization in the CaaS cloud by achieving load balancing and minimizing migration. They have investigated container migration and placement techniques for developing the constrained optimization problem. The proposed BACP algorithm for placement and ATCM algorithm for migration solve the optimization problem. An optimal container migration model has been presented in [23] by considering latency, downtime, and dependencies in the edge cloud environment. Table 1 present the detailed review on VM sizing techniques.

Motivation

VM sizing is a method to optimize the allocation of resources within the allotted spaces. Efficient VM sizing plays an important role in improving the overall performance of a CaaS cloud system. Generally, a physical machine consumes more resources than a virtual machine. To increase the overall system performance and resource utilization, containers with the packed application need to be placed in a virtual machine that requires fewer resources than a physical machine [7]. Hence, it is essential to allocate resources for a VM so that the container is executed without any wastage of resources. The ability to run multiple operating systems (OS) at the same time while utilizing different components of the hardware is the primary advantage of using virtual machines (VM). However, in the absence of virtual machines (VM), processing several operating systems would require separate physical units. In addition, the virtual machine has a low overhead, the ability to scale, and a high degree of flexibility and is used for disaster recovery. Considering all of these benefits motivates us to model the research in virtual machine sizing that is being implemented in CaaS cloud systems.

Research challenges

The different challenges encountered by the prevailing techniques of VM sizing are listed below.

- The rejection rate was increased due to the lack of correlation analysis while placing the VM.
- The accurate estimation of the resources required by the VMs was difficult due to the failure in the memory and network activity.
- The ICLB system did not consider the exploration of past resource utilization by the VM to forecast the resource usage in adapting the placement algorithms.

Table 1 Review of published work on virtual machine sizing and configuration

Author(s)	Method	Objective	Remark
S.F. Piraghaj et al. [8]	Machine Learning	Modeled the VM size selection technique	It failed to reflect on predicting workload for estimating the actual container utilization.
Lu, C.T. et al. [13]	Markov Transition	Hurst exponent and Markov transition-based VM scaling	Failed to aggregate more data to enhance the strengthening of the transition matrix to improve accuracy
Sotiriadis S. et al. [14]	Elastic Search cluster	Inter-Cloud Load Balancer (ICLB) for configuring the VM	It failed to tune the method for defining thresholds based on real-time data utilization.
Guo, Y. et al. [15]	Shadow routing based approach	Modeled the Shadow algorithm for performing the VM auto-scaling	It did not find any ways to scale the resources to minimize the count of teardowns and VM boots
Alsadie D., et al. [16]	Clustering, Machine Learning	This method successfully minimized the energy utilization in cloud data centers.	It produced a high overhead with the increased number of VMs.
Derdus, K.M. et al., [17]	Neural networks, Machine Learning	Introduced the VM right-sizing IaaS multitenant public cloud for sizing the VM	It failed to use a VM allocation algorithm for estimating the peak resources needed before allocating them.
Kenga, D.M. et al., [10]	Partition Selection Approach	VM sizing in IaaS multitenant public cloud and resource usage prediction.	It was unsuccessful in seeing the deep learning approaches
Saxena, D. and Singh, A.K., [18]	Machine Learning	Online Multi-Resource Feed-forward Neural Network (OM-FNN).	Failed to minimize the percentage of power saved with the rise in data center size.

- The OM-FNN was unsuccessful in scheduling the predicted tasks on VMs when making decisions about the management of resources.
- In some methods, defining an optimum setting is a major challenge due to the different usage of resources.

System model

Consider the CaaS cloud model shown in Fig. 1, with u number of PMs is represented as $B = \{B_1, B_2, .., B_v, .., B_u\}$. Here, each PM is comprises of a set of VMs, and is represented as V. The u^{th} PM B_u contains x number of VM, and is represented as $V = \{V_1, V_2, .., V_w, .., V_x\}$. Moreover, each PM contains distinct containers, and is indicated as L, where $L = \{L_1, L_2, .., L_s, .., L_t\}$. Moreover, the task executes in s^{th} container is represented as $U = \{U_1, U_2, .., U_s, .., U_z\}$, where z specifies the overall task count.

CaaS cloud model

The CaaS cloud model is presented in Fig. 2, where host OS present on top of the infrastructure, and the hypervisor is running in host OS. There are multiple VMs hosted in a single host machine and more than one containers can be deployed in a single virtual machine. All virtual machines are completely isolated from each other but containers in a particular VM shares same operating system [24, 25].

Proposed task grouping and optimized deep learning based VM sizing for hosting CaaS

Cloud computing is a novel archetype that provides access to services and resources available on the internet over distributed platforms. The main objective of this work is to implement an efficient VM sizing technique based on Task grouping and optimized deep learning for hosting CaaS [8]. The method is implemented using the following steps. Initially, the incoming task is acquired from the dataset. Once the incoming tasks are acquired, tasks are grouped using the Deep Embedding clustering (DEC) algorithm [26], based on the task parameters, such as task length, submission rate, scheduling class, priority, resource usage, and task latency, and task rejection rate. The task groups thus formed are denoted by G1, G2, and G3. After task grouping, VM sizing is established by using the Deep Convolutional LSTM Network (Deep-ConvLSTM) [27, 28] based on the task parameters of the groups. The Deep-ConvLSTM is tuned with the introduced Fractional Pelican Optimization (FPO) Algorithm. The introduced FPO algorithm is developed by modifying the Pelican Optimization Algorithm (POA) [29] in accordance with the Fractional calculus (FC) [30]. Once the VM requirement is determined, horizontal and vertical sizing of the VM is performed, where horizontal sizing refers to the number of containers and vertical sizing denotes the number of CPUs required. Finally, the scaling setup is reconfigured. Figure 3, depicts the schematic representation of the developed VM sizing technique

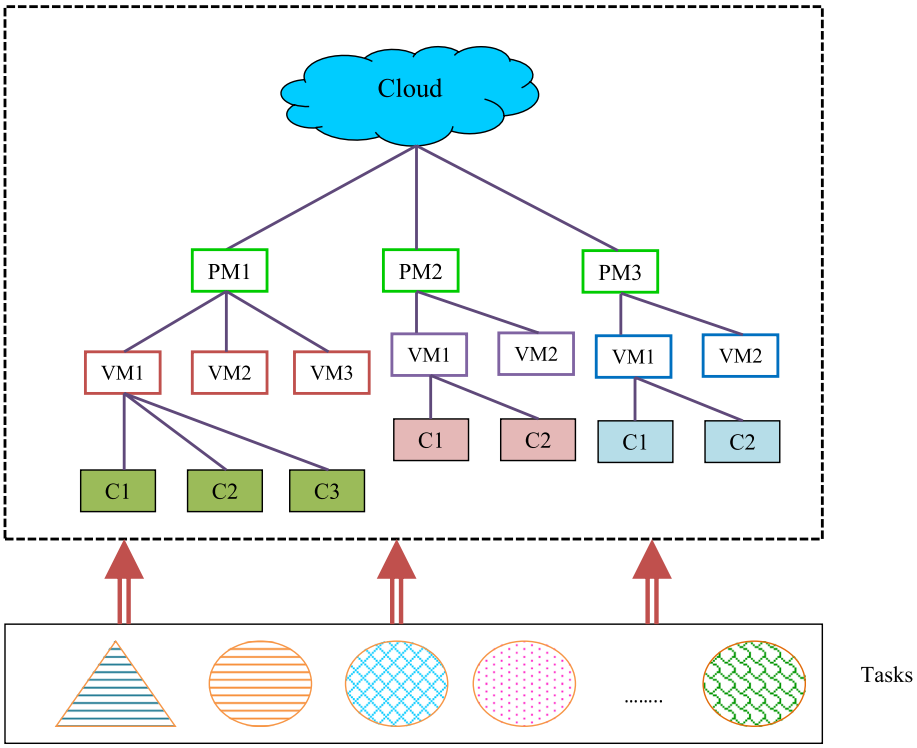

Fig. 1 CaaS Cloud System Model

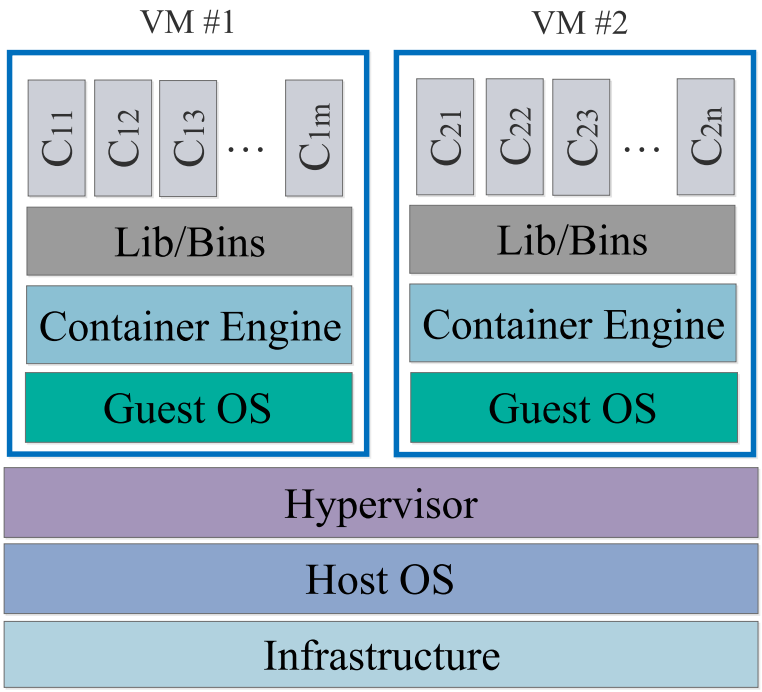

Fig. 2 Container as a Service Model

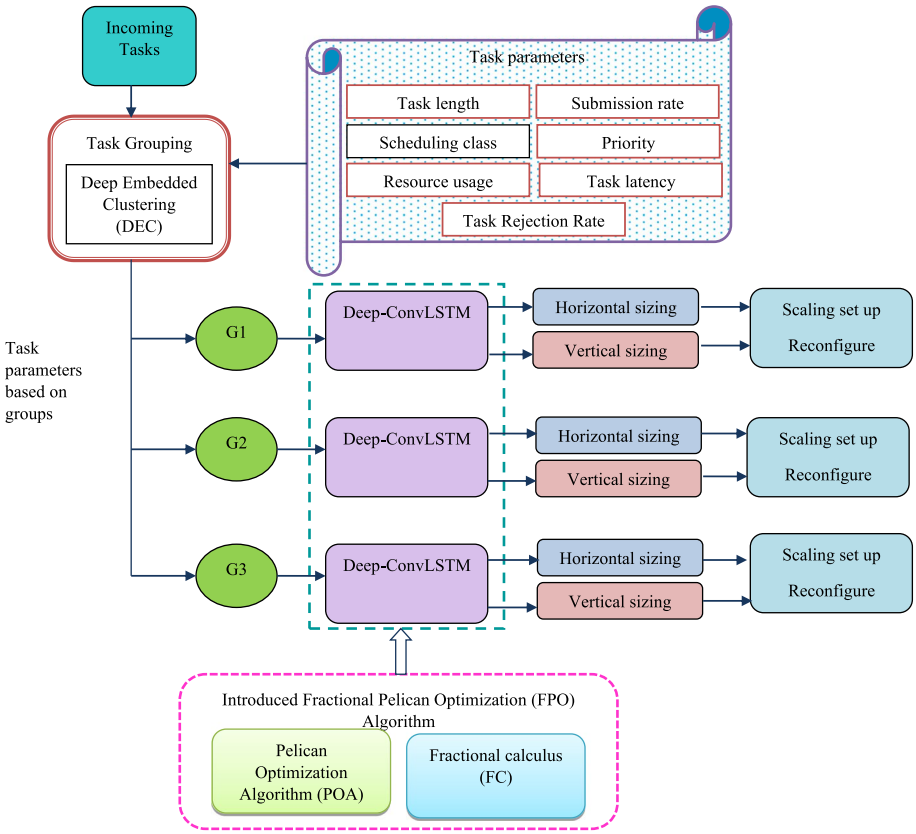

Fig. 3 Schematic representation of the developed VM sizing technique based on Task grouping and optimized deep learning for hosting CaaS

based on Task grouping and optimized deep learning for hosting CaaS.

Incoming tasks

The number of incoming task to the system is mathematically derived as,

$$G = \{G_1, G_2, ..., G_v\} \tag{1}$$

where, G indicates the total number of incoming tasks. Here, the incoming task G is given to the input of task grouping.

Task grouping using DEC

The incoming task is directed to the task grouping process in the task grouping. Here, the processing is carried out using DEC, where the grouping is done based on certain task parameters, such as task length, submission rate, scheduling class, priority, resource usage, task latency, and task rejection rate. In this research, the task grouping process groups the entire incoming task into three groups, such as D_1, D_2, and D_3, based on the task parameters. The explanation for DEC is explained in the coming sub-section.

Deep embedded clustering

DEC [26] comprises two phases, such as pretraining and clustering process for performing the task grouping. The pretraining process is performed to train the low dimensional embedded illustrations through the auto-encoder. The second phase is the clustering phase where the decoder is initially discarded and then the encoder is trained to optimize the joining of embedded illustration as well as clustering centers. For each iteration, the soft clustering assignment b_{mh} is calculated with respect to the student's t-distribution. Here, b_{mh} shows the assessment amongst embedded data point p_m and centre α_h . Let us assume the cluster centre as $\{\alpha_1, \alpha_2, ..., \alpha_n\}$, latent feature as p , then the student's t-distribution computes the similarity amongst cluster centre α_h and data points p_m, which is expressed as,

$$b_{mh} = \frac{\left(1+ \parallel p_m - \alpha_h \parallel^2\right)^{-1}}{\sum_n \left(1+ \parallel p_m - \alpha_n \parallel^2\right)^{-1}} \tag{2}$$

where, Student's t-distribution freedom degree is indicated as 1. The probability of allocating data points p_m to cluster α_h is indicated as b_{mh}, $argmax_h b_{mh}$ is represented as assigned cluster. Moreover, the clustering algorithm

continually modifies the clusters based on the learning of high confidence assignments. To learn from the assignment of high confidence, an auxiliary target distribution a_{mh} is denoted as,

$$a_{mh} = \frac{b_{mh}^2/d_h}{\sum_n b_{mh}^2/d_n} \tag{3}$$

Moreover, the KL-divergence among b_{mh} and a_{mh} learns the higher confidence soft cluster assignment, then the clustering loss is portrayed as,

$$KL(S \parallel Q) = \sum_m \sum_h a_{mh} \log \frac{a_{mh}}{b_{mh}} \tag{4}$$

where, the cluster center is indicated as m, n specifies the cluster count, d_h and d_n specifies the soft cluster frequencies. Moreover, the task grouping is done by considering several task parameters, such as task length, submission rate, scheduling class, priority, resource usage, task latency and task rejection rate, which are explained as below.

Task parameters

The task parameters used for performing the task grouping based on DEC is explained in this section. Thus, the task parameters are task length, submission rate, scheduling class, priority, resource usage, task latency, and task rejection rate, which are given below.

Task Length: Task length is the time taken for executing the task in a machine, which is represented as T_l.

Submission Rate: Submission rate is defined as how much times the task have been submitted to the data center. The submission rate is indicated as S_r.

Scheduling Class: The sensitive to latency is referred to as task/job. Here, the scheduling class is indicated by an integer among 0 and 3. The scheduling class '0' depicts the non-production task. The higher scheduling class is the greatest latency sensitive task. Moreover, the scheduling class is indicated as S_c.

Priority: The priority of task is decided based on the importance of task. The maximum preference is given to the higher priority task when compared with low priority task. Moreover, the priority is an integer, which lies among 0 to 10, and is portrayed as P.

Resource usage: The average resource usage R_U of task is derived based on memory, CPU and disk during the observed time, which is indicated as,

$$R_U = \frac{\sum_{w=1}^{c} r(U, w)}{c} \tag{5}$$

where, c indicates the task usage count in observable period. The resource usage is depicted as R_U.

Task latency: Task latency is the proportion of overall waiting time of task to the total task count, which is derived as,

$$T_l = \frac{W_t}{n_t} \tag{6}$$

where, W_t depicts the overall waiting time of task and n_t denotes the overall task count. Then, the task latency is depicted as T_l.

Task Rejection Rate (TRR): It describes the ratio of rejected task since they cannot be ordered within the time limit [31], which is described as,

$$T_r = \frac{r_t}{n_t} \tag{7}$$

where, r_t signifies the failure task count and n_t denotes the overall task. Moreover, the TRR is specified as T_r.

Furthermore, the task grouping is done based on the used task parameters, which is given by,

$$F_{min} = \frac{1}{7}[T_l + S_r + S_c + (1 - P)] + R_U + T_l + T_r \tag{8}$$

Based on the task parameters, the task grouping is done based on DEC. Once the grouping is done, then the partitioned task groups are indicated as D_1, D_2, and D_3. After the completion of task grouping, then the VM sizing is done with Deep-ConvLSTM based on the task parameters. Here, the evaluated task parameters are used to determine the amount of task for configuring the virtual machines.

VM sizing using Deep-ConvLSTM

VM sizing is a method that computes the resource allocation for VM within a physical machine. The resources include CPU, memory allocation and so on. This section explains the process of VM sizing using Deep-ConvLSTM by considering the task parameters as input. In task grouping, the task groups, such as D_1, D_2, and D_3 are obtained based on the task parameters, and is processed under the horizontal sizing and vertical sizing using Deep-ConvLSTM in order to obtain the quantity of container and quantity of CPU size. Moreover, in each group, VM sizing is done using Deep-ConvLSTM based on the group parameters. Figure 4 shows the process of VM sizing using Deep-ConvLSTM.

From Fig. 3, in VM sizing, the horizontal and vertical sizing is applies on each groups. Here, the horizontal sizing is carried out to predict the number of containers, whereas the vertical sizing is performed to predict the number of CPU size. Moreover, the architecture of Deep-ConvLSTM is explained in the below section.

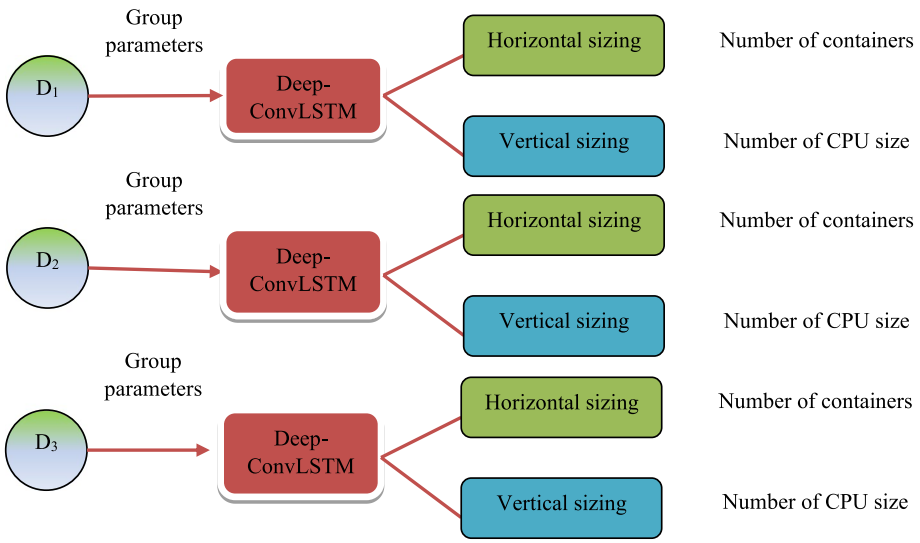

Fig. 4 VM sizing using Deep-ConvLSTM

Structure of Deep-ConvLSTM

The Deep-ConvLSTM [27, 28] is comprised of the combination of recurrent layers of LSTM and convolutional layers. The Deep-ConvLSTM is designed to train the representation of sparse features for modeling the temporal dependencies among every activation of feature illustration. Here, the convolutional layer behaves as a feature extractor in representing the extracted feature into feature maps. Likewise, the recurrent layer is employed to model the temporal dynamics of mapped features. The structure of Deep-ConvLSTM is given in Fig. 5. In Deep-ConvLSTM, the convolution layer does not perform the pooling operation. Here, the group parameters are subjected to the input layer that contains the convolutional layer, which excavates the spatial illustration of input parameters. Here, the convolutional layer computes properly only when the kernel and input fully overlap. Thus, the feature map length is computed by,

$$H^{(c+1)} = H^c - W^c + 1 \tag{9}$$

where, W^c signifies the kernel length in layer c . The conv layers utilizes the rectified linear units (ReLUs) to calculate the feature maps, whose non-linear function is portrayed as,

$$h_t^{(i+1)}(\lambda) = \chi\left(e_t^i + \sum_{f=1}^{Q^i}\left[\sum_{W=1}^{W^i}\left[k_{tf}^i(W)y_f^i(\lambda - W)\right]\right]\right) \tag{10}$$

where, $y_f^i(\lambda)$ designates the feature map in layer 1, χ be the non-linear function, Q^i specifies the feature map count, W^i denotes the kernel length, e^i be the bias vector. Thus, the output produced by the Deep-ConvLSTM is predicted container by performing the horizontal sizing and CPU size by performing the vertical sizing. Moreover, the outcome of Deep-ConvLSTM is indicated as G_k.

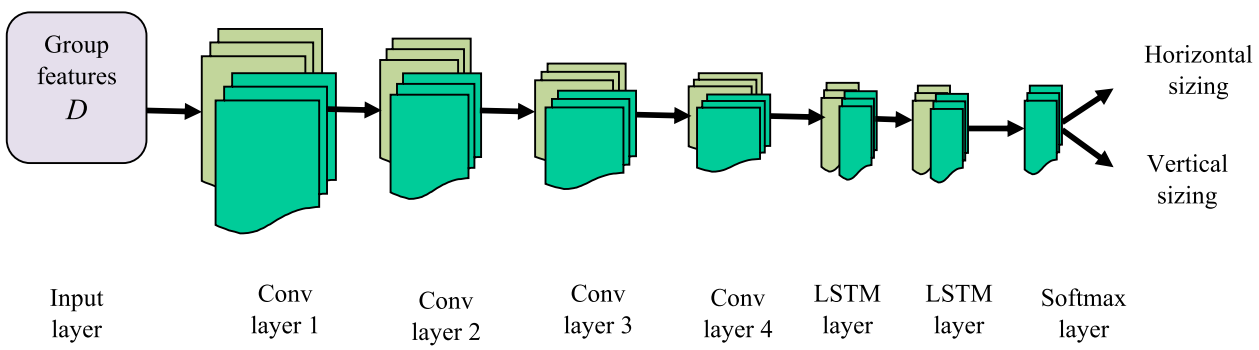

Fig. 5 Structure of Deep-ConvLSTM

Training of Deep-ConvLSTM using devised FPO

This section portrays the training process of Deep-ConvLSTM using devised FPO. Here, the newly invented FPO is the integration of FC [30] and POA [29]. POA is a swarm-based optimization approach that mimics the hunting tactics of Pelicans. The pelican has long beak along with its throat, which is used to intake the prey. The pelicans like to live in a group, and they prefer to eat fishes, turtles, frogs and sea foods. While hunting, the pelicans work together for catching the food. Moreover, the pelicans search their food at various locations. In addition, the pelicans dive from at a distance of 10 to 20m after recognizing the location of prey in order to catch the fish. In some cases, the prey may descent from it, and then they spread their wings at the posterior of water to pull the fish so that they can get the food easily. Furthermore, excess amount of water inhales into the beak bag while catching the food, and then the excess water can be drained by extending their neck outward before swallowing the food. This strategy is used for the POA, which provides the better optimization outcome, but the convergence rate of this scheme was low. The FC is the generalization of the integer order calculus (IOC), which useful in solving complex or real order problems. Engineers are interested in the Laplace and Fourier classical transforms, which are useful for solving FC differential equations as well as for simplifying procedures like convolution. The FC is integrated with optimization algorithms for effective performance. In order to improve the convergence rate and effective computation, the theory of FC is applied on the exploration behavior of POA such that the better outcome is attained. Thus, the devised FPO acquired the better convergence rate, less computational complexity and high processing speed. Thus, the algorithmic processes of invented FPO algorithm are explained as below.

1 **Initialization** In POA [29], the pelicans are considered as members of population, which are initialized in the initialization step. Moreover, each member in the population is considered as a candidate solution. Here, the population members are arbitrarily initialized based on the lower and upper bound issues, which is given by,

$$d_{k,l} = w_l + M(r_l - w_l), \qquad k = 1, 2, ..., Z \qquad l = 1, 2, ..., \chi \tag{11}$$

where, $d_{k,l}$ specifies the l^{th} variable value represented by k^{th} candidate solution, Z specifies the population member count, χ specifies the problem variable count, M indicates the random variable, w_i specifies the l^{th} lower bound, and r_l indicates the l^{th} upper bound.

2 **Fitness Computation** The importance of fitness computation is to predict the optimal solution. In this research, the number of containers and CPU size is predicted using Deep-ConvLSTM, which is trained by FPO in order to attain the optimal outcome. Moreover, the optimal outcome is attained based on the fitness function using Mean squared Error (MSE). In addition, the least value of MSE is considered as an optimal solution, and is portrayed as,

$$B_j = \frac{1}{j}\sum_{k=1}^{j}\left[G_k^* - G_k\right]^2 \tag{12}$$

where, j specifies the overall sample count, G_k specifies the predicted outcome of Deep-ConvLSTM and G_k^* indicates the expected outcome. In addition, the hunting tactics of pelicans includes exploration as well as exploitation phase. In the exploration phase, the pelicans are moving towards the prey, whereas in the exploitation phase, the pelicans are flying on the surface of water.

3 **Exploration Phase** In this step, the pelicans are searching the food by scanning the water surface, and then it moves nearer to the food after finding the prey location. In POA, the prey location is created arbitrarily in the search space, which progresses the exploration rate of optimization algorithm. Then, the mathematical derivation for moving the prey towards the food source is given by,

$$d_{k,l}^{p1} = \begin{cases} d_{k,l}(o) + M\left(J_l - N.d_{k,l}(o)\right), & B_j < B_k \\ d_{k,l}(o) + M.\left(d_{k,l}(o) - J_l\right), & \text{otherwise.} \end{cases} \tag{13}$$

where, $d_{k,l}^{p1}$ specifies the new position of k^{th} pelican in l^{th} dimension in phase 1, N indicates the random integer, J_l be the prey position in l^{th} dimension and B_j specifies the objective function. In this research, condition (1) from equation (13) is considered for finding the optimal location, which is given below. When $B_j < B_k$, then the updated expression becomes,

$$\begin{aligned} d_{k,l}(o+1) &= d_{k,l}(o) + M\left(J_l - N.d_{k,l}(o)\right) \\ &= d_{k,l}(o) + M.J_l - M.N.d_{k,l}(o) \\ &= d_{k,l}(o)(1 - M.N) + M.J_l \end{aligned} \tag{14}$$

Subtract $d_{k,l}(o)$ on both sides,

$$d_{k,l}(o+1) - d_{k,l}(o) = d_{k,l}(o)(1 - M.N) + M.J_l - d_{k,l}(o) \tag{15}$$

In order to upsurge the prediction performance of devised scheme, the concept of FC is applied on the updated location of POA. From FC,

$$R^{\varepsilon}\left[d_{k,l}(o+1)\right] = d_{k,l}(o)(1 - M.N - 1) + M.J_l \tag{16}$$

$$d_{k,l}(o+1) - \varepsilon.d_{k,l}(o) - \frac{1}{2}d_{k,l}(o-1) - \frac{1}{6}(1-\varepsilon)d_{k,l}(o-2) - \frac{1}{24}\varepsilon(1-\varepsilon)(2-\varepsilon)d_{k,l}(o-3) = d_{k,l}(o)(-M.N) + M.J_l \quad (17)$$

$$\begin{aligned} d_{k,l}(o+1) =& d_{k,l}(o)(-M.N) + M.J_l + \varepsilon.d_{k,l}(o) + \frac{1}{2}d_{k,l}(o-1) \\ &+ \frac{1}{6}(1-\varepsilon)d_{k,l}(o-2) + \frac{1}{24}\varepsilon(1-\varepsilon)(2-\varepsilon)d_{k,l}(o-3) \end{aligned} \quad (18)$$

where, $d_{k,l}(o+1)$ depicts the location of k^{th} solution in l^{th} dimension at iteration $0+1$, $d_{k,l}(o-1)$ depicts the location of k^{th} solution in l^{th} dimension at iteration $o-1$, $d_{k,l}(o-2)$ depicts the location of k^{th} solution in l^{th} dimension at iteration $o-2$, $d_{k,l}(o-3)$ depicts the location of k^{th} solution in l^{th} dimension at iteration $o-3$, J_l denotes the position of prey in l^{th} dimension, M signifies the random variable and N indicates the parameter, which is equal to 1 or 2.

4 **Exploitation Phase** In this step, the pelicans spread their wings when it comes nearer to the surface of water such that the fish move upwards, then the pelicans gathered their food in the mouth. In this tactic, large amount of food is gotten by the pelicans. Then, the hunting aspects of pelicans are referred to as,

$$d_{k,l}^{p2}(o+1) = d_{k,j}(o) + E.\left(1 - \frac{a}{T}\right).(2.M-1).d_{k,l}(o) \quad (19)$$

where, $d_{k,l}^{p2}$ signifies the new location of k^{th} pelican in l^{th} dimension in phase 2, T indicates the maximum iteration count, a specifies the iteration counter and $\left(1 - \frac{a}{T}\right)$ depicts the nearest radius of $d_{k,l}$. Moreover, the effective updated solution is utilized to remove or accept the new position of pelicans, and is articulated as,

$$D_k = \begin{cases} D_k^{p2}, & B_k^{p2} < B_k \\ D_k, & \text{otherwise.} \end{cases} \quad (20)$$

where, D_k^{p2} shows the new location of k^{th} pelican, and B_k^{p2} indicates the objective function of phase 2.

5 **Re-evaluation of Feasibility** Here, the feasibility is evaluated based on the fitness function. The fitness function is computed for all iteration. If the predicted fitness is better than the previous one, then the devised FPO replaces the older solution with newer one such that the optimal location is updated at all iteration.

6 **Termination** The steps from (i) to (v) is repeated till the optimal solution is attained. Moreover, the pseudocode of invented FPO algorithm is explained in Algorithm 1. Here, the optimal predicted solution is attained using Deep-ConvLSTM-based FPO, where the FPO is configured by the joining of POA and FC, such that the better solution is acquired.

Input: $CandidateSolutions$.
Output: $OptimalSolution$.
Introduce the optimization constraints.
Calculate the population size z and iteration number T.
Set the location of pelicans by equation (14).
Calculate the objective function.

```
for o ← 1 to T do
    Produce the prey locality at random manner
    for N ← 1 to z do
        Do exploration phase
        for l ← 1 to s do
            Compute the new position of l^th dimension by equation (13)
        end for
        Replace the k^th population number by equation (18)
        Perform exploration phase
        for l ← 1 to x do
            Calculate the new station of l^th dimension by equation (19)
        end for
        Replace the k^th population member by equation (20)
    end for
    Re-compute the best candidate solution
end for
```

Algorithm 1 FPO Algorithm

Scaling setup and reconfigure

After the completion of VM sizing, then the scaling setup and reconfiguration process is initialized based on the number of containers and number of CPU size such that the overall system is reconfigured.

Results and discussion

The results and discussion of devised Deep-ConvLSTM_FPO is deliberated in this section. Moreover, the utilized dataset, metrics and implementation tool is explained in this section.

The devised Deep-ConvLSTM_FPO scheme is implemented in Java with CloudSim tool having PC with Windows 10 OS and intel i3 core processor. The dataset used for the Deep-ConvLSTM_FPO scheme is Google trace dataset [32]. The Google trace dataset contains various fields, such as job ID, task index, cache memory usage, machine ID, average CPU usage rate, maximum CPU usage and so on. The performance metrics used for devised task grouping and VM sizing are task capacity, task rejection rate, and resource utilization. Response time is the time taken to give the first response.

Performance analysis

The performance of the devised Deep-ConvLSTM_FPO is evaluated with various population sizes by varying the iteration, and the results are discussed in this section. Here, the evaluation is carried out by adjusting the task sizes, such as 100, 200, 300, and 400.

Table 2 Performance Assessment of Deep-ConvLSTM_FPO with Task Size 100

Number of Iteration	Deep-ConvLSTM_FPO with Population Size 5	Deep-ConvLSTM_FPO with Population Size 10	Deep-ConvLSTM_FPO with Population Size 15	Deep-ConvLSTM_FPO with Population Size 20
Resource Utilization				
5	0.0275	0.0258	0.0226	0.0207
10	0.0365	0.0326	0.0308	0.0287
15	0.0375	0.0358	0.0326	0.0316
20	0.0355	0.0315	0.0287	0.0258
Response Time				
5	277	275	272	270
10	269	265	262	260
15	263	260	258	254
20	262	259	256	253
Task Rejection Rate				
5	0.1565	0.1477	0.1368	0.1257
10	0.1659	0.1579	0.1479	0.1379
15	0.1875	0.1755	0.1654	0.1579
20	0.1987	0.1868	0.1766	0.1654
Makespan				
5	0.6368	0.6258	0.6079	0.5975
10	0.6146	0.6077	0.5877	0.5765
15	0.5968	0.5868	0.5654	0.5534
20	0.5789	0.5679	0.5468	0.5357

Performance assessment based on task size 100

Table 2 shows the performance analysis of the Deep-ConvLSTM_FPO scheme based on the resource utilization, response time, TRR, and Makespan using the task size as 100. Here, the analysis is done based on varying the iterations as 5, 10, 15, and 20, and the performance is maximum at iteration=20. The FPO is configured by combining POA and FC to achieve better performance.

Performance assessment based on task size 200

The performance analysis of the Deep-ConvLSTM_FPO scheme based on the resource utilization, response time, TRR, and makespan using the task size of 200 is shown in Table 3. The iteration considered for the evaluation is 5, 10, 15, and 20, and the proposed model's population size is 5, 10, 15, and 20. The performance of the Deep-ConvLSTM_FPO scheme is improved when increasing the number of iterations.

Performance assessment based on task size 300

Table 4 presents the analysis of the Deep-ConvLSTM_FPO scheme based on resource utilization, response time, TRR, and makespan using task size 300. Considering the task size 300, the maximum performance offered by the Deep-ConvLSTM_FPO is 0.0936 for resource utilization, 472 ms for response time, 0.3088 for Task Rejection Rate, and 0.6368 for makespan.

Performance assessment based on task size 400

The analysis of the Deep-ConvLSTM_FPO scheme based on the resource utilization, response time, TRR, and makespan using the task size 400 is depicted in Table 5. From this table, it is clear that the proposed Deep-ConvLSTM_FPO offers maximum results when considering the iteration is 20 and the population size is 20.

Comparative methods

The various traditional techniques used for comparing the performance of Deep-ConvLSTM_FPO scheme are VM size selection technique [8], Hurst exponent +Markov transition based VM scaling [13], ICLB [14], VM size IaaS multi-tenant public cloud [17], OM-FNN [18], Deep-ConvLSTM GD, and Deep-ConvLSTM ADAM.

Comparative analysis

The comparison of Deep-ConvLSTM_FPO scheme with existing methods is analyzed by adjusting the task sizes, such as 100, 200, 300 and 400.

Table 3 Performance Assessment of Deep-ConvLSTM_FPO with Task Size 200

Number of Iteration	Deep-ConvLSTM_FPO with Population Size 5	Deep-ConvLSTM_FPO with Population Size 10	Deep-ConvLSTM_FPO with Population Size 15	Deep-ConvLSTM_FPO with Population Size 20
Resource Utilization				
5	0.0654	0.0636	0.0588	0.0547
10	0.0733	0.0715	0.0677	0.0636
15	0.0714	0.0687	0.0636	0.0598
20	0.0721	0.0699	0.0647	0.0615
Response Time				
5	540	538	534	531
10	449	446	443	440
15	442	440	439	436
20	427	422	419	416
Task Rejection Rate				
5	0.2054	0.1979	0.1876	0.1757
10	0.2241	0.2168	0.2088	0.1977
15	0.2385	0.2258	0.2157	0.2076
20	0.2486	0.2368	0.2279	0.2168
Makespan				
5	0.5656	0.5543	0.5368	0.5146
10	0.5866	0.5754	0.5543	0.5368
15	0.6077	0.5865	0.5754	0.5543
20	0.6357	0.6268	0.6157	0.6075

Table 4 Performance Assessment of Deep-ConvLSTM_FPO with Task Size 300

Number of Iteration	Deep-ConvLSTM_FPO with Population Size 5	Deep-ConvLSTM_FPO with Population Size 10	Deep-ConvLSTM_FPO with Population Size 15	Deep-ConvLSTM_FPO with Population Size 20
Resource Utilization				
5	0.0987	0.0958	0.0925	0.0908
10	0.1025	0.0995	0.0976	0.0943
15	0.1025	0.0987	0.0965	0.0937
20	0.1041	0.0999	0.0975	0.0936
Response Time				
5	650	649	646	643
10	527	525	522	520
15	511	508	503	501
20	482	478	475	472
Task Rejection Rate				
5	0.3254	0.3168	0.3088	0.2987
10	0.3303	0.3268	0.3168	0.3077
15	0.3401	0.3368	0.3268	0.3168
20	0.3387	0.3258	0.3168	0.3088
Makespan				
5	0.6088	0.5968	0.5765	0.5645
10	0.6257	0.6146	0.6079	0.5975
15	0.6579	0.6466	0.6257	0.6146
20	0.6755	0.6644	0.6457	0.6368

Table 5 Performance Assessment of Deep-ConvLSTM_FPO with Task Size 400

Number of Iteration	Deep-ConvLSTM_FPO with Population Size 5	Deep-ConvLSTM_FPO with Population Size 10	Deep-ConvLSTM_FPO with Population Size 15	Deep-ConvLSTM_FPO with Population Size 20
Resource Utilization				
5	0.1279	0.1198	0.1165	0.1147
10	0.1325	0.1301	0.1287	0.1257
15	0.1479	0.1458	0.1425	0.1401
20	0.1488	0.1465	0.1436	0.1418
Response Time				
5	634	630	627	624
10	697	694	691	688
15	619	617	615	612
20	693	690	688	683
Task Rejection Rate				
5	0.3855	0.3755	0.3654	0.3579
10	0.3959	0.3865	0.3765	0.3666
15	0.4014	0.3977	0.3865	0.3765
20	0.4113	0.4077	0.3968	0.3865
Makespan				
5	0.6257	0.6146	0.5976	0.5865
10	0.6579	0.6643	0.6543	0.6468
15	0.6977	0.6865	0.6757	0.6643
20	0.7157	0.7087	0.6977	0.6864

Comparative analysis based on task size 100

Table 6, shows the analysis of the Deep-ConvLSTM_FPO scheme based on the resource utilization, response time, TRR, and Makespan using the task size as 100. Here, the analysis is done based on varying the iterations as 5, 10, 15, and 20, and the maximum performance is observed with 20 iterations. The performance of the Deep-ConvLSTM_FPO acquired better outcomes because of the better convergence rate, less computational complexity, and high processing speed of the devised FPO.

Comparative analysis based on task size 200

The analysis of the Deep-ConvLSTM_FPO scheme based on the resource utilization, response time, TRR, and Makespan using the task size of 200 is shown in Table 7. The iteration considered for the evaluation is 5, 10, 15, and 20. The performance of all the models is improved when increasing the iterations, which the Deep-ConvLSTM_FPO attains maximum results than other conventional methods.

Comparative analysis based on task size 300

Table 8, presents the analysis of Deep-ConvLSTM_FPO scheme based on resource utilization, response time, TRR, and Makespan using the task size 300. Considering the task size 300, the maximum performance offered by the Deep-ConvLSTM_FPO is 0.1041 for resource utilization, 482ms for response time, and 0.3387 for Task Rejection Rate.

Comparative analysis based on task size 400

The analysis of the Deep-ConvLSTM_FPO scheme based on the resource utilization, response time, TRR, Makespan using the task size 400 is depicted in Table 9. From this table, it is clear that the proposed Deep-ConvLSTM_FPO offers maximum results than other existing methods, such as VM size selection technique, Hurst exponent+Markov transition , ICLB, OM-FNN, VM size IaaS multi-tenant public cloud, Deep-ConvLSTM GD, and Deep-ConvLSTM ADAM.

Performance assessment based on makespan

We also compared the proposed method with all existing methods by considering makespan. Compared to existing approaches, the time it takes to complete a set of tasks utilizing the proposed Deep-ConvLSTM FPO is significantly shorter. The performance of the proposed model using makespan is shown in Fig. 6.

Discussion

Table 10 shows the comparative discussion of the proposed Deep-ConvLSTM_FPO for task grouping and

Table 6 Comparative Assessment of Deep-ConvLSTM_FPO Scheme With Task Size 100

Number of Iteration	VM size selection technique	Hurst exponent+Markov transition	ICLB	OM-FNN	VM size IaaS multi-tenant public cloud	Deep-ConvLSTM GD	Deep-ConvLSTM ADAM	Proposed Deep-ConvLSTM_FPO
Resource Utilization								
5	0.0397	0.0381	0.0325	0.0291	0.0289	0.0286	0.0283	0.0275
10	0.0567	0.0407	0.0406	0.0387	0.0377	0.0374	0.0370	0.0365
15	0.0498	0.0454	0.0454	0.0408	0.0400	0.0387	0.0381	0.0375
20	0.0451	0.0430	0.0405	0.0399	0.0397	0.0380	0.0369	0.0355
Response Time								
5	18188	13184	12420	8456	6225	892	567	277
10	17784	14780	11985	8578	6193	945	599	269
15	17801	15798	11980	8635	6118	987	634	263
20	17758	16755	11982	8743	6129	1034	678	262
Task Rejection Rate								
5	0.2055	0.1978	0.1848	0.1808	0.1785	0.1765	0.1728	0.1565
10	0.2214	0.2055	0.1985	0.1848	0.1794	0.1764	0.1737	0.1659
15	0.2413	0.2254	0.2054	0.2016	0.1990	0.1958	0.1937	0.1875
20	0.2855	0.2541	0.2325	0.2257	0.2143	0.2137	0.2110	0.1987
Makespan								
5	0.7876	0.7578	0.7145	0.6965	0.6876	0.6754	0.6543	0.6367
10	0.7976	0.7755	0.7356	0.7154	0.7078	0.6865	0.6754	0.6145
15	0.8156	0.7987	0.7467	0.7357	0.7267	0.7076	0.6875	0.5967
20	0.8478	0.8257	0.7765	0.7578	0.7476	0.7245	0.7087	0.5788

Table 7 Comparative Assessment of Deep-ConvLSTM_FPO Scheme With Task Size 200

Number of Iteration	VM size selection technique	Hurst exponent+Markov transition	ICLB	OM-FNN	VM size IaaS multi-tenant public cloud	Deep-ConvLSTM GD	Deep-ConvLSTM ADAM	Proposed Deep-ConvLSTM_FPO
Resource Utilization								
5	0.0733	0.0726	0.0725	0.0708	0.0683	0.0674	0.0669	0.0654
10	0.0792	0.0777	0.0764	0.0760	0.0758	0.0756	0.0749	0.0733
15	0.0944	0.0794	0.0779	0.0764	0.0757	0.0747	0.0737	0.0714
20	0.1040	0.0916	0.0893	0.0826	0.0793	0.0765	0.0758	0.0721
Response Time								
5	34856	28852	23345	15678	11846	921	621	540
10	36799	30796	25355	15710	12805	956	658	449
15	35340	31337	23719	15846	11891	999	692	442
20	36451	32448	24747	15937	11815	1025	734	427
Task Rejection Rate								
5	0.2999	0.2945	0.2914	0.2901	0.2895	0.2755	0.2698	0.2054
10	0.3587	0.3413	0.3321	0.3157	0.3026	0.2968	0.2877	0.2241
15	0.3699	0.3585	0.3486	0.3268	0.3143	0.3077	0.2976	0.2385
20	0.3413	0.3399	0.3325	0.3326	0.3326	0.3269	0.3179	0.2486
Makespan								
5	0.6867	0.6467	0.6076	0.5976	0.5865	0.5778	0.5725	0.5656
10	0.7076	0.6578	0.6145	0.6096	0.6076	0.5967	0.5935	0.5865
15	0.7156	0.6765	0.6567	0.6467	0.6356	0.6256	0.6156	0.6076
20	0.7267	0.6976	0.6677	0.6576	0.6478	0.6436	0.6400	0.6356

Table 8 Comparative Assessment of Deep-ConvLSTM_FPO Scheme With Task Size 300

Number of Iteration	VM size selection technique	Hurst exponent+Markov transition	ICLB	OM-FNN	VM size IaaS multi-tenant public cloud	Deep-ConvLSTM GD	Deep-ConvLSTM ADAM	Proposed Deep-ConvLSTM_FPO
Resource Utilization								
5	0.1428	0.1223	0.1095	0.1058	0.1040	0.0997	0.0993	0.0987
10	0.1338	0.1286	0.1220	0.1187	0.1133	0.1120	0.1101	0.1025
15	0.1233	0.1195	0.1175	0.1164	0.1158	0.1137	0.1117	0.1025
20	0.1495	0.1217	0.1185	0.1181	0.1179	0.1147	0.1127	0.1041
Response Time								
5	51830	42142	34677	20456	17531	981	734	650
10	51791	43258	34657	20583	17500	1021	786	527
15	51872	45125	34719	20725	17493	1089	848	511
20	52590	47853	35405	20892	17849	1156	891	482
Task Rejection Rate								
5	0.4013	0.3954	0.3658	0.3569	0.3414	0.3368	0.3318	0.3254
10	0.4125	0.3999	0.3714	0.3688	0.3515	0.3486	0.3458	0.3303
15	0.4185	0.4014	0.3799	0.3627	0.3699	0.3647	0.3618	0.3401
20	0.4014	0.3985	0.3854	0.3800	0.3479	0.3447	0.3418	0.3387
Makespan								
5	0.7689	0.7245	0.6865	0.6578	0.6367	0.6257	0.6145	0.6087
10	0.7866	0.7367	0.7076	0.6867	0.6578	0.6468	0.6367	0.6256
15	0.8087	0.7578	0.7146	0.6967	0.6798	0.6678	0.6624	0.6578
20	0.8256	0.7790	0.7468	0.7156	0.6865	0.6846	0.6825	0.6755

Table 9 Comparative Assessment of Deep-ConvLSTM_FPO Scheme With Task Size 400

Number of Iteration	VM size selection technique	Hurst exponent+Markov transition	ICLB	OM-FNN	VM size IaaS multi-tenant public cloud	Deep-ConvLSTM GD	Deep-ConvLSTM ADAM	Proposed Deep-ConvLSTM_FPO
Resource Utilization								
5	0.1522	0.1460	0.1347	0.1331	0.1328	0.1312	0.1301	0.1279
10	0.1842	0.1675	0.1525	0.1494	0.1476	0.1437	0.1401	0.1325
15	0.1798	0.1714	0.1568	0.1537	0.1519	0.1501	0.1500	0.1479
20	0.1963	0.1748	0.1619	0.1595	0.1568	0.1537	0.1517	0.1488
Response Time								
5	68881	58878	46085	40123	23119	15457	981	634
10	68910	61906	46115	40475	23292	15843	1036	697
15	68765	62762	45951	40734	23171	16284	1091	619
20	68984	63981	46207	40982	23250	16790	1158	693
Task Rejection Rate								
5	0.4585	0.4413	0.4326	0.4235	0.4126	0.4087	0.4037	0.3855
10	0.4659	0.4585	0.4488	0.4458	0.4325	0.4268	0.4179	0.3959
15	0.4854	0.4785	0.4585	0.4529	0.4458	0.4379	0.4268	0.4014
20	0.4986	0.4876	0.4785	0.4654	0.4587	0.4479	0.4368	0.4113
Makespan								
5	0.8765	0.8578	0.8236	0.7967	0.7865	0.6654	0.6467	0.6256
10	0.8578	0.8467	0.7976	0.7867	0.7755	0.6976	0.6654	0.6578
15	0.8367	0.8076	0.7765	0.7654	0.7578	0.7367	0.7245	0.6976
20	0.8087	0.7865	0.7578	0.7467	0.7367	0.7268	0.7226	0.7156

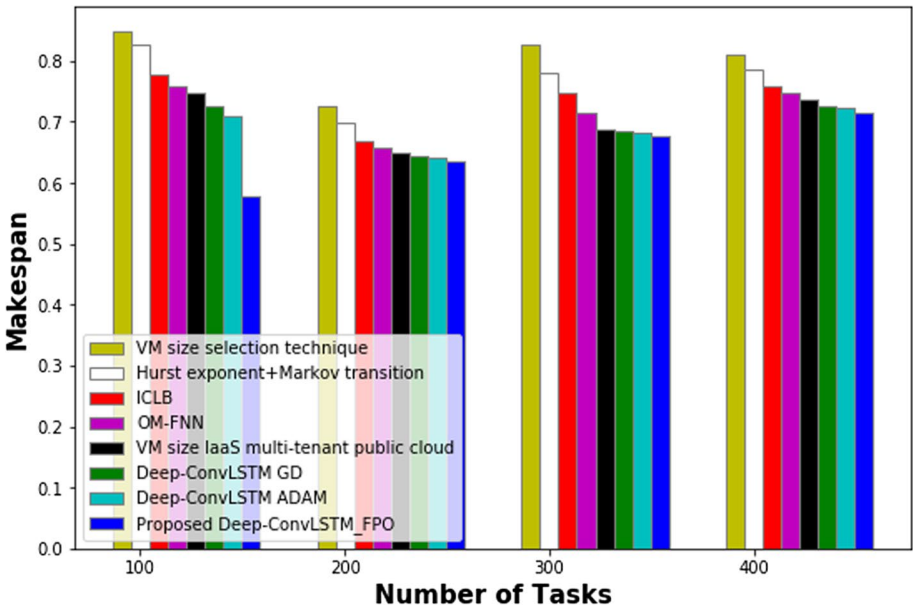

Fig. 6 Observed Makespan Using Different Approaches

VM sizing. Here, the proposed method is evaluated with four different task sizes, such as 100, 200, 300, and 400. Moreover, the assessment is done by considering different metrics, such as resource utilization, response time, and TRR. From Table 10, it is observed that the developed method attained better values than the comparative methods. In this research, the proposed scheme achieved better performance than other approaches due to the accuracy of task grouping. The grouping of tasks is done based on several effective task parameters, such as

Table 10 Comparative Discussion

Metrics	VM size selection technique	Hurst exponent+Markov transition	ICLB	OM-FNN	VM size IaaS multi-tenant public cloud	Deep-ConvLSTM GD	Deep-ConvLSTM ADAM	Proposed Deep-ConvLSTM_FPO
100 Tasks								
Resource Utilization	0.0451	0.0430	0.0405	0.0399	0.0397	0.0380	0.0369	0.0355
Response Time (ms)	17758	16755	11982	8743	6129	1034	678	262
TRR	0.2855	0.2541	0.2325	0.2257	0.2143	0.2137	0.2110	0.1987
Makespan	0.8478	0.8257	0.7765	0.7578	0.7476	0.7245	0.7087	0.5788
200 Tasks								
Resource Utilization	0.1040	0.0916	0.0893	0.0826	0.0793	0.0765	0.0758	0.0721
Response Time(ms)	36451	32448	24747	15937	11815	1025	734	427
TRR	0.3413	0.3399	0.3325	0.3326	0.3326	0.3269	0.3179	0.2486
Makespan	0.7267	0.6976	0.6677	0.6576	0.6478	0.6436	0.6400	0.6356
300 Tasks								
Resource Utilization	0.1495	0.1217	0.1185	0.1181	0.1179	0.1147	0.1127	0.1041
Response Time(ms)	52590	47853	35405	20892	17849	1156	891	482
TRR	0.4014	0.3985	0.3854	0.3800	0.3479	0.3447	0.3418	0.3387
Makespan	0.8256	0.7790	0.7468	0.7156	0.6865	0.6846	0.6825	0.6755
400 Tasks								
Resource Utilization	0.1963	0.1748	0.1619	0.1595	0.1568	0.1537	0.1517	0.1488
Response Time(ms)	68984	63981	46207	40982	23250	16790	1158	693
TRR	0.4986	0.4876	0.4785	0.4654	0.4587	0.4479	0.4368	0.4113
Makespan	0.8087	0.7865	0.7578	0.7467	0.7367	0.7268	0.7226	0.7156

task length, submission rate, scheduling class, priority, resource usage, task latency, and TRR. An efficient VM sizing can increase resource utilization, reduce response time, and minimize TRR.

Threat to validity

Even though we made every effort to ensure that our experiment was carried out correctly, there is still a chance that it was flawed in a number of different ways. When interpreting or immediately applying the findings or conclusions presented in this study, future efforts should consider these limitations and account for them.

The extent to which inferences made about the discovered ideas are accurate is known as construct validity. One danger that fits into this category is that we tend to group multiple ideas under the umbrella of a single term. The problem of construct validity arises whenever insufficient measurement of the variables is employed. The construct of the experiment, its design, and its trustworthiness can threaten the results. A semantic mistake in the firewall policy or an incorrect packet delivery on the networks might negatively impact the results. To mitigate the impact of the threat, we decided to experiment with some guidelines that were actually released publicly.

The term "internal validity" describes the extent to which findings about the connection between examined resources and data gathered may be categorized as casual. The term "maturation", which refers to a negative effect, such as tiredness, on participants while participating in an experiment, poses a threat to the study's internal validity. During the execution of Deep-ConvLSTM, it was our responsibility to carry out operations that needed us to perform the same actions many more than once. For example, we had to cluster the data set into three groups and acquire the input data.

The extent to which conclusions that are only true on an internal level can be generalized is known as the study of external validity. The phrase "interaction of context and therapy" is a good illustration of this type of danger. The setting to which the results should be applied is not the same as our Deep-ConvLSTM uses. Hence the two settings cannot be compared. As a result, the community being addressed (cloud engineers, cloud customers, researchers, and cloud architects) has varying expertise compared to the field that the sources that were reviewed originated from. To protect ourselves against this danger, we have modified our search keywords to include the term "cloud computing," expecting that this will make the search results better for the intended audience.

Conclusion

This paper devises the newly modeled task grouping and VM sizing approach, Deep-ConvLSTM FPO. Here, the devised FPO algorithm is modeled by incorporating FC with POA, which is used to train the weights of Deep-ConvLSTM by considering MSE as a fitness value. In this research, the VM sizing is carried out with Deep-ConvLSTM by applying it separately to each group obtained by the task grouping. Moreover, horizontal sizing and vertical sizing are applied to each group, where horizontal sizing is performed to predict the number of containers. In contrast, vertical sizing is performed to predict CPU sizes. The task grouping is done by DEC, which considers several task parameters, such as task length, submission rate, scheduling class, priority, resource usage, task latency, and TRR. Furthermore, the devised scheme reconfigures the entire system after completing the horizontal and vertical sizing. In addition, the performance of VM sizing is done based on the metrics such as utilization, response time, TRR, and makespan of 0.035, 262 ms, and 0.198, and 0.5788 correspondingly. In the future, the performance can be further enhanced by introducing an excess quantity of task parameters. Also, dynamic task scheduling will be considered in the further extension of the proposed method.

Author' contributions

Manoj Kumar Patra: Conceptualization, Methodology, Writing- Original draft preparation. Bibhudatta Sahoo: Data curation, Supervision. Ashok Kumar Turuk: Methodology, Supervision. Sanjay Misra: Investigation, Validation, Reviewing and finalising the paper. The author(s) read and approved the final manuscript.

Funding

No funding available.

Availability of data and materials

We have used Google Traces Data sets which is publicly available and can be downloaded from, https://research.google/tools/datasets/google-cluster-workload-traces-2019/.

Declarations

Ethics approval and consent to participate

Not applicable.

Competing interests

The authors declare no competing interests.

Received: 22 August 2022 Accepted: 11 April 2023
Published online: 25 April 2023

References

1. Kumar P, Kumar R (2019) Issues and challenges of load balancing techniques in cloud computing: A survey. ACM Comput Surv (CSUR) 51(6):1–35

2. Cloud H (2011) The nist definition of cloud computing, vol 800. National Institute of Science and Technology, Special Publication, pp 145
3. Subramanian N, Jeyaraj A (2018) Recent security challenges in cloud computing. Comput Electr Eng 71:28–42
4. Malomo O, Rawat DB, Garuba M (2018) A survey on recent advances in cloud computing security. J Next Gener Inf Technol 9(1):32–48
5. Hussein MK, Mousa MH, Alqarni MA (2019) A placement architecture for a container as a service (caas) in a cloud environment. J Cloud Comput 8(1):1–15
6. Boukadi K, Rekik M, Bernabe JB, Lloret J (2020) Container description ontology for caas. Int J Web Grid Serv 16(4):341–363
7. Zhang R, Chen Y, Zhang F, Tian F, Dong B (2020) Be good neighbors: A novel application isolation metric used to optimize the initial container placement in caas. IEEE Access 8:178195–178207
8. Piraghaj SF, Dastjerdi AV, Calheiros RN, Buyya R (2015) Efficient virtual machine sizing for hosting containers as a service (services 2015). In: 2015 IEEE World Congress on Services, IEEE, pp 31–38
9. Zhang R, Chen Y, Dong B, Tian F, Zheng Q (2019) A genetic algorithm-based energy-efficient container placement strategy in caas. IEEE Access 7:121360–121373
10. Kenga DM, Omwenga VO, Ogao PJ (2019) Autonomous virtual machine sizing and resource usage prediction for efficient resource utilization in multi-tenant public cloud. Int J Inf Technol Comput Sci(IJITCS) 11(5):11–22
11. Meng X, Isci C, Kephart J, Zhang L, Bouillet E, Pendarakis D (2010) Efficient resource provisioning in compute clouds via vm multiplexing. In Proceedings of the 7th international conference on Autonomic computing, pp 11–20
12. Jahani A, Lattuada M, Ciavotta M, Ardagna D, Amaldi E, Zhang L (2019) Optimizing on-demand GPUs in the Cloud for Deep Learning Applications Training 2019, 4th International Conference on Computing, Communications and Security (ICCCS), Rome, pp 1–8. https://doi.org/10.1109/CCCS.2019.8888151
13. Lu CT, Chang CW, Li JS (2015) VM scaling based on Hurst exponent and Markov transition with empirical cloud data. J Syst Softw 99:199–207. https://doi.org/10.1016/j.jss.2014.10.011
14. Sotiriadis S, Bessis N, Amza C, Buyya R (2019) Elastic Load Balancing for Dynamic Virtual Machine Reconfiguration Based on Vertical and Horizontal Scaling, vol 12. In: IEEE Transactions on Services Computing, (no. 2), pp 319–334. https://doi.org/10.1109/TSC.2016.2634024
15. Guo Y, Stolyar AL, Walid A (2020) Online VM Auto-Scaling Algorithms for Application Hosting in a Cloud, vol 8. In: IEEE Transactions on Cloud Computing, (no. 3), pp 889–898. https://doi.org/10.1109/TCC.2018.2830793
16. Alsadie D, Tari Z, Alzahrani EJ, Zomaya AY (2018) Dynamic resource allocation for an energy efficient VM architecture for cloud computing. In: Proceedings of the Australasian Computer Science Week Multiconference (ACSW '18). Association for Computing Machinery, New York, Article 16, pp 1–8. https://doi.org/10.1145/3167918.3167952
17. Derdus K, Omwenga V, Ogao P (2019) Virtual machine sizing in virtualized public cloud data centres. Int J Sci Res Comput Sci Eng Inf Technol 5(4). https://doi.org/10.32628/CSEIT1953124
18. Saxena D, Singh AK (2021) A proactive autoscaling and energy-efficient VM allocation framework using online multi-resource neural network for cloud data center. Neurocomputing 426:248–264
19. Piraghaj SF (2016) Energy-efficient management of resources in container-based clouds. PhD thesis, Ph. D. dissertation, University of Melbourne, Australia
20. Liu J, Wang S, Zhou A, Xu J, Yang F (2020) Sla-driven container consolidation with usage prediction for green cloud computing. Front Comput Sci 14(1):42–52
21. Liagkou V, Fragiadakis G, Filiopoulou E, Michalakelis C, Kamalakis T, Nikolaidou M (2022) A pricing model for container-as-a-service, based on hedonic indices. Simul Model Pract Theory 115:102441
22. Zhang W, Chen L, Luo J, Liu J. A two-stage container management in the cloud for optimizing the load balancing and migration cost. Future Generation Comput Syst 135(2022):303–314
23. Aleyadeh S, Moubayed A, Heidari P, Shami A (2022) Optimal container migration/re-instantiation in hybrid computing environments. IEEE Open J Commun Soc 3:15–30
24. Patel D, Patra MK, Sahoo B (2020) Gwo based task allocation for load balancing in containerized cloud. In: 2020 International Conference on Inventive Computation Technologies (ICICT), IEEE, pp 655–659
25. Patra MK, Patel D, Sahoo B, Turuk AK (2020) Game theoretic task allocation to reduce energy consumption in containerized cloud. In: 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence), IEEE, pp 427–432
26. Xie J, Girshick R, Farhadi A (2016) Unsupervised deep embedding for clustering analysis. In: International conference on machine learning, PMLR, pp 478–487
27. Ordóñez FJ, Roggen D (2016) Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. Sensors 16(1):115
28. Lim XY, Gan KB, Abd Aziz NA (2021) Deep convlstm network with dataset resampling for upper body activity recognition using minimal number of imu sensors. Appl Sci 11(8):3543
29. Trojovskỳ P, Dehghani M (2022) Pelican optimization algorithm: A novel nature-inspired algorithm for engineering applications. Sensors 22(3):855
30. Bhaladhare PR, Jinwala DC (2014) A clustering approach for the-diversity model in privacy preserving data mining using fractional calculus-bacterial foraging optimization algorithm. Adv Comput Eng 2014:12
31. Marahatta A, Wang Y, Zhang F, Sangaiah AK, Tyagi SKS, Liu Z (2019) Energy-aware fault-tolerant dynamic task scheduling scheme for virtualized cloud data centers. Mob Netw Appl 24(3):1063–1077
32. Datasets GT (2019) Clusterdata 2019 traces. https://research.google/tools/datasets/google-cluster-workload-traces-2019/. Accessed 6/7/2022

Publisher's Note