Computer Networks 185 (2021) 107684

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

The two-phase scheduling based on deep learning in the Internet of Things

Shabnam Shadroo, Amir Masoud Rahmani *, Ali Rezaee

Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran

ARTICLE INFO

Keywords:
Fog-cloud computing
Autoencoder
Task scheduling

ABSTRACT

With the growth of data generation speed and its huge volume, the cloud-based infrastructure alone does not meet the needs of the Internet of Things (IoT), thereby leading to inefficiency. By combining fog and cloud computing on the IoT, some processing occurs near the data generation location with a higher speed and without needing a large bandwidth. Cloud and fog computing requires proper management in scheduling, increasing resource efficiency, and reducing consumption costs. In this article, a two-phase scheduling algorithm is presented based on deep learning methods in the field of IoT. The first phase is to decide on the location of the task execution using the clustering method. The second phase involves scheduling the task according to the execution location. In the clustering section, three ideas based on the Self-Organizing Map (SOM) clustering method have been proposed. In the first and second ideas, the SOM and hierarchical SOM are used to cluster the features of the tasks received from the IoT layer. In the third idea, the feature is extracted, and its dimensions are reduced using the Autoencoder, which is one of the deep learning methods, after which clustering is done. After scheduling each cluster's tasks, comparing the methods presented in the article, it is shown that the feature extraction using deep learning can improve clustering in such a way to reduce the missed rate of tasks in the cloud and fog, as well as their costs.

1. Introduction

With the emergence of any technology, society will either admit it during the period of growth and maturity or forget it because of the emergence of newer technology or its failure to meet target communities' needs. In the meantime, the capability of a new technology to interact with existing technologies, especially covering their problems, can significantly impact the technology's penetration and growth coefficient.

For example, applying IoT has been far faster than traditional methods when presented to the human community. IoT is used in many applications such as industrial control, intelligent transportation, and smart medicine, creating copious amounts of data. Cloud computing methods have come alongside IoT to store, process, and analyze huge amounts of data [1–4].

By integrating IoT with the cloud, the stream of data storage and processing has become easier, thereby accelerating installation and integration and reducing the cost for data processing and their complex performance [5].

However, the use of cloud presents new challenges that traditional cloud computing architecture cannot solve. By applying the cloud, time-sensitive applications will face problems, where cloud computing does not provide mobility and location-awareness, and security requirements. To overcome these problems, a new issue, called "fog", was introduced by Cisco in 2012 to solve many of these problems [6].

The fog is the computing layer close to the receiving layer and provides computing, networking, and storage services [6]. It provides both fog and cloud services. The cloud differs from the fog in decentralizing, processing large amounts of local data, installing heterogeneous software, proximity to the end-user, geographical distribution density and mobility, and security support. Fig. 1 displays the fog environment. The fog is suitable for scenarios where many decentralized and heterogeneous devices are interconnected and require collaboration, storing, and processing tasks to be a practical method for IoT [10].

By combining cloud and fog, methods were developed to manage these platforms together to enhance efficiency. These methods have tried to distribute tasks to reduce the network traffic and the workload on the central networks and cloud servers. In some of these methods, the IoT nodes are clustered at the edge of the network based on the resources type, the required time, etc. to process the requested task in parallel and distributed. To cluster the tasks and then schedule them, [7,8] used the Kmean method, applied Fuzzy clustering, [9] utilized Mixed Integer

* Corresponding author.
E-mail address: rahmani@srbiau.ac.ir (A.M. Rahmani).

https://doi.org/10.1016/j.comnet.2020.107684
Received 29 June 2020; Received in revised form 6 October 2020; Accepted 11 November 2020
Available online 15 November 2020

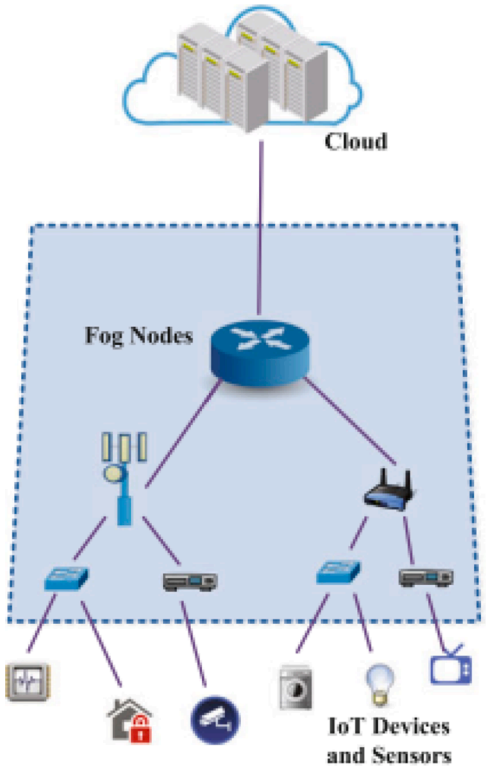


Fig. 1. Fog computing environment [6].

Linear Programming, [10] used game theory, [11] employed K-medoid, and [12] applied Meta-heuristic algorithm.

The devices on the edge of the network send their requests to the fog and cloud. These requests can be reviewed on the fog, where according to the task type, they are sent to fog or cloud upon request. In this article, the tasks are first received from the IoT layer. The features of the operating system type, task type, task priority, task privacy, task data heterogeneity, task execution, task arrival time, task resource, task data, and maximum required task data of the received tasks are obtained to enter the first phase (clustering section). Then, in the clustering section, they are clustered according to the task features. To cluster tasks, three methods have been proposed, which are based on SOM. In the first two methods, clustering is performed on the initial features obtained from the tasks using SOM and hierarchical SOM. The third method uses one of the deep learning methods to extract the features. The Autoencoder networks are considered unsupervised. In this method, these networks are used to extract and reduce features. First, the features are extracted by the Autoencoder, and then the SOM clusters them. After clustering the tasks, we will schedule the task according to the next phase's intended cluster. After scheduling, the three proposed methods in the article are compared with each other, whereby it is determined that the use of Autoencoder in task clustering leads us to achieve the main objectives of this article, which are reducing the rate of missed tasks and the costs of using fog as well as cloud including bandwidth, memory, and processing.

The rest of the article is organized as follows. The second section reviews the literature on the Internet of Things, big data, cloud computing, fog computing, and deep learning. The basic concepts will be overviewed in the third section. Two-phase scheduling will be described in the fourth section. In the fifth section, we review the results and compare them with other methods. Finally, the sixth section concludes the paper.

2. Literature review

The concept of IoT was innovated by Kevin Ashton in 1999, exactly when the idea of a broader device-to-device connection came true. IoT is defined as a self-organizing system of unrestricted devices that provides convergence systems to improve process efficiency. Further, it provides a reference for object identification on the Internet; IoT allows all kinds of communication and data sharing. It offers various services through the connection of virtual objects and the interoperability of information technology [13].

Cloud computing is on-demand Internet-based computing, which includes payment for each usage of applications and resources. Considering the benefits of high computing power, low service costs, better performance, scalability, accessibility, and availability, cloud computing is becoming a requirement in the IT industry. Cloud services are provided over the Internet, and the devices which want to access cloud services must have internet access. In this case, the devices will require a smaller memory, a very simple operating system, and a browser [5].

By increasing the connectivity of different devices on IoT, time-sensitive applications may encounter some problems. Also, cloud computing does not provide support for mobility and location awareness. To overcome this problem, a new issue, called fog, was presented in 2012. A cloud computing environment consists of traditional network components such as routers, switches, workstations, and proxy servers adjacent to IoT devices and sensors. Bonomi et al. proposed the fog as a powerful virtual platform that provides storage, computing, and networking environments between the cloud data center and end-devices. Both the fog and cloud provide data, computing, storage, and application services for the end-user; however, their differences are in decentralization, processing of large amounts of local data, installation of heterogeneous software and hardware, proximity to the end-user, geographical distribution density, and provision of mobility. The fog is suitable for scenarios where many decentralized and heterogeneous devices are interconnected and require collaboration, storage, and processing tasks. Further, the users can check their fog at any time using any device connected to the network [14].

In recent years, many IoT applications have emerged in various domains, including health, transportation, smart homes, smart cities, agriculture, education, etc. Many of these applications' main elements are intelligent learning mechanisms for prediction (i.e., regression, classification, and clustering), data mining and pattern recognition, or data analysis. Deep Learning (DL) has been actively used in many IoT applications in recent years, among many machine learning approaches. These technologies (DL and IoT) are among the top three technology majors of 2017 announced at the Gartner / ITxpo 2016 Symposium. The reason for these intensive advertisements for DL arises from the fact that traditional machine learning approaches do not meet the new analytical needs of IoT systems. Instead, IoT systems require analytical data approaches and artificial intelligence approaches appropriate to the hierarchy of data generation and IoT data management, as illustrated in Fig. 2 [15].

Deep learning is a special model of machine learning. Unlike traditional machine learning algorithms, which break down a problem into smaller parts and solve them individually, deep learning uses the end-to-end method to solve the problem. We have compared deep learning with traditional machine learning in feature learning, model construction, model training, data requirement, and training time. This comparison is shown in Table 1.

In machine learning, the operation begins with the manual extraction of features. Then, using feature selection methods, the appropriate feature is selected to construct a model that does the categorization. In machine learning, each model is built step by step and usually has a shallow structure. Extracting and selecting the appropriate feature is very time-consuming and depends on the designer's knowledge. In deep learning, the feature extraction process is not manual, and along with

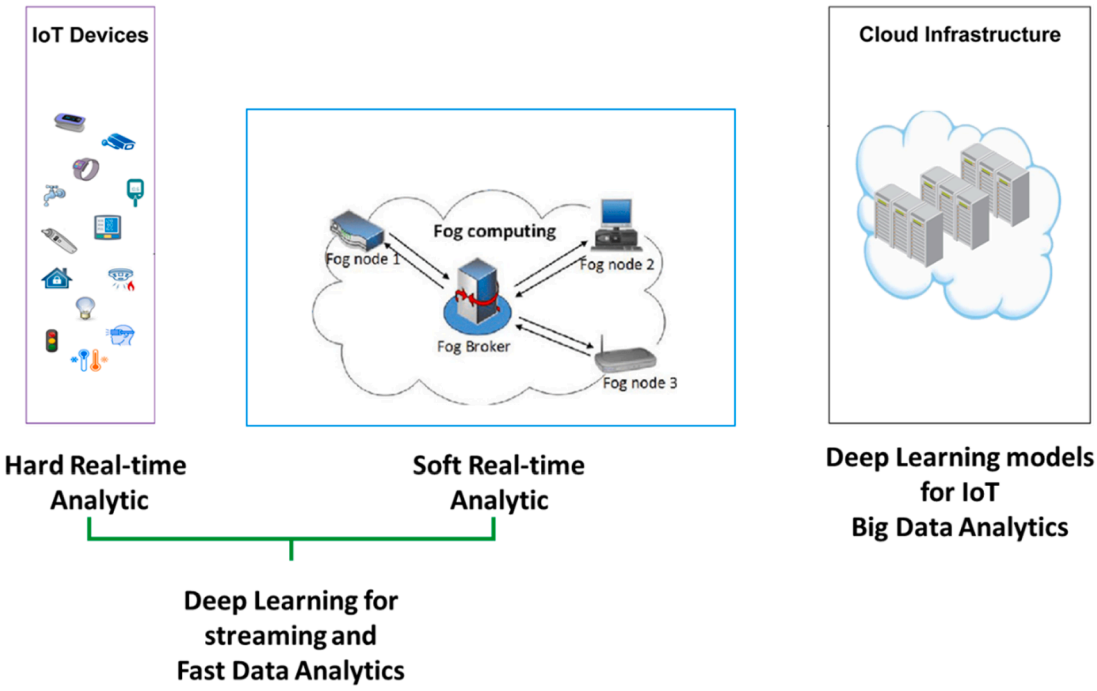


Fig. 2. Generating IoT Data at Different Levels and DL Samples [15].

Table 1
Comparison of Traditional Machine Learning and Deep Learning.

	Traditional Machine Learning	Deep Learning
Feature Learning	Using the feature extraction method with human intervention	Automatic feature extraction
Model Construction	Applying extracted features to construct a data-driven model, with shallow structures	An end-to-end high hierarchical model structure with nonlinear combining of multi-layers
Model training	Training each module step-by-step	Jointly training the parameters
Data requirement	Requiring large data	Lesser data are trained
Training Time	Long time	Less time

model construction, are performed by selecting different kernels and setting parameters through end-end optimization. Applying many hidden layers and deep architecture leads to multi-level nonlinear operations. The transfer of each layer representation from the main input to a more abstract representation in the higher layers leads to finding the data's inherent structure. These abstracted feature representations are then considered as the input of the next layer for classification or regression. The end-end learning structure reduces human intervention, where the deep learning parameters are continuously trained. Deep learning algorithms require large data, so they do not work properly when the data is small. In deep learning, the model expands with increasing data, but in machine learning, the model expands to a certain extent. One of the key features of deep learning is that it improves with increasing data. Training deep learning algorithms takes considerable time since many parameters have to be checked and trained to the system, while this time is far shorter in machine learning. Finally, although machine learning improves its performance by consuming less cost and less data, it still needs expert guidance and human intelligence. However, deep learning can improve itself through a deep neural network and be trained by its previous calculation method. It will also further contribute to the goal of artificial intelligence knowledge, which is designing a system with the greatest human-like intelligence [16,17].

Deep learning methods have been implemented in different architectures as follows:

- Convolutional deep neural networks
- Recurrent neural networks
- Autoencoder neural networks
- Deep belief networks
- Generative adversarial networks

Autoencoders are a family of neural networks that are well suited for unsupervised learning. They are a means for detecting the inherent pattern in datasets and appropriate for labeling the obtained patterns. Autoencoders essentially reconstruct datasets and, during a process, discover their inherent structure and extract its important features. Table 2 compares some of the deep learning approaches. There are various deep learning tools, the most important ones being Torch, Tensor Flow, Caffe, Theano, and DL4J [15].

Table 3 outlines several products that use deep learning in their smart cores and can serve IoT domains in fog or cloud. In this section, we have studied IoT articles. The first section deals with articles focusing on

Table 2
Comparison of Deep Learning Implementation Techniques[17].

Features	Convolutional Neural Network	Restricted Boltzmann Machine	Auto Encoder
Generalization capability	Yes	Yes	Yes
Unsupervised learning	No	Yes	Yes
Feature Learning	Yes	Yes	Yes
Real-time training	No	No	Yes
Training with low number datasets	Yes	Yes	Yes

Table 3
The Products Applying Deep Learning in Their Platform [15].

Product	Description	Application	Platform
Amazon Alexa	IPA (Intelligent Personal Assistant)	Smart Home	Fog
Microsoft Cortana	IPA	Smart Machine, XBOX	Fog
Google Assistant	IPA	Smart Machine, Smart Home	Fog
IBM Watson	Cognitive framework	IoT domains	Cloud

IoT, using the cloud, fog, and sometimes big data tools, and then we will examine articles on the scheduling domain.

2.1. Reviewing IoT articles in cloud-fog computing from the big data point of view

Behera et al. [18] applied an IoT architecture to use the cloud for data storage, processing, and analysis. After extracting useful information, this information was sent to another cloud, provided to the users after the final analysis. They used this architecture in a smart power management system.

Sowe et al. [19] presented an IoT architecture on the big data platform whose purpose was to manage services and heterogeneous and complicated sensor communications. The big data platform helped the data be easily identified and managed by different sensors and share their knowledge and air pollution experience.

Paul et al. [20] introduced an intelligent body architecture capable of receiving useful information from a smart city and monitoring household consumption. This architecture comprised three domains, the object, the Social Internet of Things (SIoT), and the program. Different data were collected from heterogeneous devices and sent to the server in the SIOT layer for storage and processing in the object layer. The data were pre-processed in the SIOT layer and then distributed to the servers, whereby offline and online data analysis was performed. This architecture used Hadoop for data analysis as well as MapReduce and Spark for real-time offline processing. Elsewhere, Páez et al. proposed architecture for monitoring chronic patients and caring for individuals, outdoors and indoors, and related services. The most important part of this architecture included smart mobile devices, message platforms, website platforms, and a health information module. In this architecture, the information received from the individual was stored on a mobile device. After reaching a certain volume, it was stored in a private cloud service, whereby the data could be processed in near real-time. The Riak database, which is distributed and scalable, was used in this architecture. It had many different sensors, and using this organization, the query could provide the latest information of any sensor within a short time. The authors focused more on data storage rather than its analysis [21]. Another study reviewed an architecture for cloud-based health software that supported both IoT and big data. The architecture's layers included a reception layer, transmission layer, and healthcare cloud layer. The healthcare cloud layer was divided into two sub-layers, including the cloud service and the cloud service plan. The cloud service layer's main task was to collect physical data from the physical world and human data compressed, formatted, stored, and analyzed. This layer is the main support of the IoT top health services and application. The cloud application layer was the interface between the public health network and users, which directly represented the economic and social estimates generated by IoT. It was responsible for controlling all sensor nodes, representing the visual data, and capturing the work and business flow [22].

Suciu et al. [23] used the remote machine-to-machine measurement architecture with thgrape industry's big data platform. The system included M2M, CloudIOT, and big data processing. Another article examined the existing components and methods of securing big data processing with M2M cloud systems based on remote measurement. They presented their proposed architecture for smart health applications [24,25].

In [26], the data stream processing and online and offline analyses were performed, which were scalable and fault-tolerant based on the cloud using the Ahab framework. Vögler et al. used this framework in which online analysis and pattern learning from the stored data were allowed in the Smart City scenario.

Khorshed et al. first examined the safety of three-layer models, addressed the challenges and issues of security and technology. They then tried to attack the three-layer architecture, including the data collection layer, reception layer, and analysis layer. This research tried to use the objects layer, cloud layer, and big data layer in the data collection layer. The big data layer had attempted to settle the Hadoop ecosystem in the cloud [27]. The embedded sensor networks could generate a significant amount of data transmitted or transferred to a remote cloud server after being collected. Since the sensors had limited capacity, they had to discard old data or stop producing new products to discharge old data. Mozumdar et al. expressed that this data could reach one gigabyte in a week. A limited-capacity orchestrator could handle the space required for a small sensor network for a short period; thus, data management and storage capacity for large-scale applications was needed. The use of the Rame-Douglas-Pevcker compression algorithm on sensor data yielded 99.86% data compression [28].

Dineshkumar et al. [29] stored the data received from the cloud sensors and used them to monitor patients' health, where the data included all patients' information. In this study, the analysis was used in the cloud by HDFS and Mapreduce, where the proposed system had different sensors comparing to other systems and preformed real-time analysis.

Manogaran et al. [30] presented an architecture for health care application, which consisted of two parts called Meta Fog-R direction (MF-R) and Choosing & Grouping (CG). In the MF-R section, electronic drug records were collected through clinical examination where their results were stored in the cloud. When the data was transferred from applications to cloud data centers, it should be effectively stored. In this architecture, HDFS was used to store and construct a model to predict the patient's condition using data history. A logistic regression algorithm was implemented via map-reduce and the Mahout library. If the patient's condition was not good, the system transmitted clinical values to the doctor via the fog computing through wireless networking. After identifying the authorized user, it safely transmitted medical data to other data centers and cloud servers. The CG section was responsible for securing the connection between fog and cloud computing.

Kholod et al. [31] introduced two different IoT architectures for the data mining system. These architectures included the receiving layer, gateway access layer, network layer, and application layer. The middleware layer had two layers of data collection and data processing in the centralized state, where the data mining layer was in the application layer. In this architecture, the cloud was used for data storage, while in the distributed architecture, the cloud and the fog node were used to converge some of the computations towards the data source. In the distributed architecture, the data collection layer was embedded in the network layer. The data processing layer was in the middleware layer stored by the cloud, while the data mining layer was in the application layer. In this article, the data mining algorithm was mapped to the functional blocks on the activator model. The activator model was designed to provide parallel and distributed computations. The parallel data mining algorithm was implemented on activators that could exchange a message with the main activator. Activators could be distributed and implemented in the cloud and fog nodes. The results presented in this article indicated that the execution time of the algorithm in IoT with distributed architecture based on the activator model was faster when dealing with the big data set than with the centralized architecture. It is because it did not require time to load data into the centralized storage. Further, guiding computations toward the center would increase the efficiency of the IoT system.

Table 4 compares the articles mentioned in this section based on the ideas, advantages, disadvantages, software used, and the evaluation criteria.

2.2. Reviewing IoT articles with deep learning

Li He et al. [32] emphasized the edge-based computing with deep learning. The proposed model was adapted to different deep learning models in the IoT edge computation. Further, an online algorithm was presented to improve the service capacity where the discharge offloading strategy was proposed to improve the performance. Due to the varying size of the middle data and the pre-processing overload in different deep learning models, the article presented a scheduling problem for maximizing the number of deep learning tasks by limiting the network bandwidth and the service capacity of the edge nodes. Then, it provided online and offline algorithms to solve the problem. In order to reduce the network traffic load when transferring data from the sensors to the cloud server, a scheduling algorithm was proposed. Deep learning had two layers (a convolutional layer and the fully connected layer); however, the question was which layer should be placed on the edge server. Edge servers were more restrictive than cloud servers, and in deep learning, as they went towards deeper layers, the data size was reduced; thus, if we could put those layers to the edge server, the network traffic would improve; however, as mentioned earlier, the edge server had some restrictions on its capacity. In this article, an algorithm for solving this problem was presented, which tried to accomplish most edge computing structure tasks by applying deep layers to the edge IoT server. The required transferring latency of each task could be guaranteed.

Peng Li et al. [33] proposed a Deep Complex Computational Model (DCCM) in the tensor space to represent any heterogeneous object. Elaraby et al. [34] presented a system based on the Autoencoder, which identified and removed lost data in the preparation section using machine learning. Afterward, the KNN identified the anomaly and was removed from the data. The SOM was used to reduce the data dimension and was sent to the Autoencoder network in the processing section.

Mohammadi et al. [35] used semi-supervised learning methods in their article and analyzed the scenario of finding users' location in smart buildings. They indicated semi-supervised deep learning methods would be better than their deeply supervised counterparts.

Yisheng et al. [36] utilized the stacked Autoencoder to learn the traffic flow features and applied a standard prediction (logistic regression) in the upper layer of the Autoencoder. The experiments indicated that the proposed method had a better performance in predicting the traffic flow.

To perform simultaneously and in real-time flexibility identification and prediction, Mocanu et al. [37] proposed a new IoT framework using deep learning methods. The proposed framework evaluated smart electric meters for ten buildings. They showed that this framework performed well in simultaneous energy analysis and identified and predicted flexibility.

Alam et al. [38] simulated and compared eight data mining algorithms using three sensor datasets in the UCI site, R software. The results indicated that C4.5 and C5.0 had good accuracy, efficient memory, and high processing speed, while ANN and DLANN had high computational complexity despite their high accuracy.

Table 5 compares the articles mentioned in this section based on the ideas, advantages, disadvantages, and software used and the evaluation criteria.

2.3. Reviewing task scheduling methods

Fang et al. [39] presented a task scheduling algorithm based on load balancing, implemented in two steps. In the first step, the virtual machine allocation of applications took place, while in the second one, the appropriate resources for the virtual machine were provided. The independence of tasks and the absence of repetitive tasks were the hypotheses of the article. Further, only the time of response and demand for resources were considered in this model, while the bandwidth and the cost were not considered.

Yin et al. [40] (2018) used containers to schedule and allocate resources. Their goal was to overcome the disadvantages of using a virtual

Table 4
The Comparison of Articles on Cloud-Fog Computing from The Big Data Point of View.

Ref	Year	Ideas	Advantages	Disadvantages	Framework/ Software
[18]	2015	Providing an effective architecture for cloud computing and big data	- Efficient data management - Reduction of data collection costs	Not considering security and quality parameters and bandwidth	Hadoop
[19]	2014	Integrated architecture combination of the service controlled network (SCN) features with the cloud	Managing services and heterogeneous sensor communications	Not supporting the mobile sensors	Not stated
[20]	2016	Integration of IoT with social networks	Investigation of human interaction with electronic devices in the IoT environment and social networks	Excluding security	Hadoop
[21]	2014	Health services based on the big data and IoT	Using Riak database	The analysis is not discussed	Not stated
[23]	2015	Using CloudIOT and big data to predict disease and alert in grape cultivation	-Using the advantages of M2M -Early detection of disease in the grape farm	Does not have smart big data and is based on some conditions	Exalead cloud view
[26]	2017	-Providing a scalable and error-tolerant framework for cloud-based data processing	Error-tolerant scalable online and offline analysis	It is not fully developed and is not responsive to machine learning techniques	Not stated
[27]	2015	-Studying security in a three-layer model and identifying cyberattacks	Considering security in big data, cloud, and IoT	Accuracy is not appropriate	Weka
[28]	2014	-Three-dimensional modeling of the analysis -Providing the flow compression algorithm	Suitable for big data analysis		Raspberry-Pi MySQL
[29]	2016	Monitoring patients' health from data stored in the cloud	The response time of the proposed method is low and is suitable for real-time tasks		Arduino Hadoop
[30]	2017	Meta Fog-Redirection Grouping & Choosing	- Using the cloud to provide a scalable storage environment -Considering security	The focus has been on storage	Mahout Hadoop
[31]	2016	Distributed implementation of data mining algorithm	- Using cloud and fog - Increasing IoT system performance - Increasing the processing speed in distributed mode	Not considering security	DXelopes

Table 5
The of Articles on the IoT and Deep Learning.

Ref	Year	Ideas	Advantages	Disadvantages	Framework /Software	Evaluation criteria
[32]	2018	- Using edge computations with CNN -Reducing the input data	- Better performance for big data - Less time to interpret information -privacy persevering	Good performance is achieved after a long period	Python Caffe	Number of applied tasks
[33]	2018	Using tensor representation for multidimensional features	- Avoiding over-fitting -Improving training efficiency - High accuracy	More time for the feature learning on the heterogeneous data	Not stated	Accuracy Training time
[34]	2017	Sensor data processing using an autoencoder	- Emphasizing more on data preparation. - Extracting more effective features through deep learning	- Adjusting the values of deep model parameters - The capacity of the DL architecture is directly related to the number of hidden units	Matlab Weka Rapid miner	Accuracy Specificity Sensitivity
[35]	2018	Using semi-supervised reinforcement deep learning	Introducing a general framework used for a variety of IoT applications		TensorFlow	Performance Accuracy
[36]	2015	Using stacked Autoencoder e to learn features from Spatio-temporal traffic flow data	Improving the traffic flow prediction performance	The prediction layer is logistic regression and not examine other ones	Not stated	Mean Absolute Error, Mean Relative Error, Root-Mean-Square Error
[37]	2016	Using a kind of RBM to predict the real-time buildings' energy	- Real-time flexibility identification and prediction simultaneously - Good performance - Good accuracy	Not considering what parameters affect the performance	Matlab	Power Normalized Root Mean Square Error (NRMSE) Time of NRMSE Accuracy
[38]	2016	Studying Data Mining Methods in IoT	Studying the effectiveness and efficiency of data mining algorithms	The datasets are not big	R	Accuracy

machine. The presented algorithm was used for real-time industrial activities. They used fog to provide a real-time response for tasks. Each fog node had three measurements, including request, task scheduler, and resource management. The task schedule section determined whether to run the tasks in the fog or the cloud. The results indicated that the proposed method reduced task delay and increased the fog node's number of concurrent tasks.

Tang et al. [41,42] presented a task scheduling model based on a genetic algorithm to optimize energy consumption. They considered the dependency of tasks, data transfer, task performance deadline, and cost in this model. In this study, each task could be uploaded to the mobile cloud and locally to the mobile device, where those tasks that required interaction with the mobile user were performed on the mobile device. Further, the task required more computing and, thus, more energy consumption was stored in the mobile cloud. The results of this scheduling indicated that it worked optimally in power consumption and response time.

Hung et al. [43] reviewed a task scheduling method to ensure better access to the cloud network and mobile cloud computing processing speed, considering the network bandwidth constraints and the cost of using the cloud. The method used had two parts; task priority and appropriate processor selection. However, details on how to obtain some criteria, such as the first start time, the end time of non-provided tasks, and the algorithm's complexity were unknown.

Chu et al. [44] discussed the task scheduling in fog computing to improve computational time. In this article, the task dependency and the interconnection between devices were considered. It was also assumed that the execution time of a task could be predicted using different types and amounts of resources. In this article, scheduling visual data in the fog was transformed into a linear scheduling problem, whereby a new solution was presented. They indicated that the proposed method was scalable and practical.

Tychala et al. [45] designed a system for fog computing as Bag-of-tasks (BOT). The Local Dispatcher (LD) section of the system considers each task as a BOT and according to the authors' routing policies and adds it to the proper subsystem. Some of the LD's policies include considering the system's lowest load, the shortest waiting queue, executing tasks in its local cluster after applying a particular threshold, etc.

Tian et al. [46] presented a method based on task clustering. The proposed method included three steps of task clustering, prioritizing, and processor allocating to them. In the clustering section, tasks with high communication overhead were placed in a cluster. Then, each task's priority was examined due to the dependence of the tasks, where a list of tasks that were ready to be performed was created. Finally, the appropriate processor was assigned to the task. The proposed method reduced the SLR values and increased the speed up values compared to the article's mentioned methods and improved performance. The proposed method reduces the task execution time and uses all available resources.

Mohammed Zaki Hasan et al. [47] proposed scheduling DDSS and h-DDSS in the IoT environment based on the heuristic method. The proposed scheduling algorithm uses CPSO (Canonical Particle Swarm Optimization) and fully informed particle swarm optimization (FIPS) algorithms to ensure Quality of Service (QoS) in survivability applications. Using these algorithms, prioritization is made between incoming requests and available cloud resources according to different data traffic classes. The authors proposed three strategies for updating particle velocities based on system utilization ratio, arrival rates, and service rate to improve convergence, stating that the proposed algorithm would converge to at least one local or global optimization. The results show that the FIPS algorithm optimizes throughput and delay parameters compared to the CPSO algorithm.

In another article, Mohammed Zaki Hasan and his colleague [48] used the CPSO algorithm in scheduling the IoT-enabled smart job-shop. They applied CPSO to select tasks to minimize the makespan, utilization ratio, and flowtime, along with various scheduling algorithms such as Longest Processing Time (LPT), Shortest Processing Time (SPT), Earliest Computation Time (ECT), Earliest Starting Time (EST), Earliest Deadline First (EDF), and Earliest Due Date (EDD). The decision-making model for scheduling is based on dynamic scheduling rules. The defined framework is a dedicated network of VMs controlled by several unrelated machines. They assumed that the processing time of tasks in VMs is different. The scheduling algorithms sort users' requests based on the

processing time, which is specified or estimated. The CPSO then evaluates the objective function, which in this article is to minimize the makespan, to select the best solution. The simulation results show that the proposed method enhances performance by decreasing the makespan and utilization ratio and increasing throughput.

Table 6 compares the articles mentioned in this section based on the ideas, advantages, disadvantages, software used, and the evaluation criteria.

3. Overview of basic concepts

3.1. Self-organizing map (SOM)

Self-Organizing Map was invented in the 1980s by Kohonen and was first used in speech recognition and, more precisely, in transforming speech into writing. Self-Organizing Map is a new method for visualizing information with high dimensionality. This method converts nonlinear and complex statistical information with high dimensionality into simple geometric relation with low dimensionality. Although this method compresses the information, the essential metric and topological relationships between information are maintained. Further, this method can be considered as a form of information abstraction. The critical features in visualization and information abstraction are applied to solve complex problems such as process analysis, machine perception, control, and communication. The SOM is usually a regular, two-dimensional network of neurons, with each neuron representing a model of observations [49].

Self-organized maps are a type of unsupervised learning-based artificial neural network which displays the samples under training in a low dimensional and categorized space. The network uses a neighborhood function to maintain the topology features of the input space. The routine of placing a vector on a map is to find the node with the closest weight vector to the data space vector [50].

The Self-Organizing Map consists of neurons in a regular low-dimensional grid, two- or three-dimensional. The number of neurons may vary from several tens to several thousand. Each neuron is assigned a d dimensional vector with m weight with d denoting the input vectors' dimension. The neurons are connected to their adjacent neurons by a neighborhood relation that affects the topology or the map [51].

SOM training is similar to the Vector Quantization algorithm, or Kmeans, with the important difference that its typology neighbors are also updated on the map in addition to the most appropriate weight vector. The result is that the neurons in the network are aligned, and the adjacent neurons receive similar weight vectors. Fig. 3 displays a model of the self-organizing map. For simplicity, elements are assumed to form an ordered flatten array as a neuron or groups of neurons interacting with each other, with each element representing a set of M_i numerical value called a model. [51].

These values may correspond to some nervous system parameters. Each model is assumed to be improved by the messages received from that element. Input x, which is a neural message and a set of parallel signal values, is spread in the M_i models set. There is a common term called "competition" between elements in the human brain; if a common

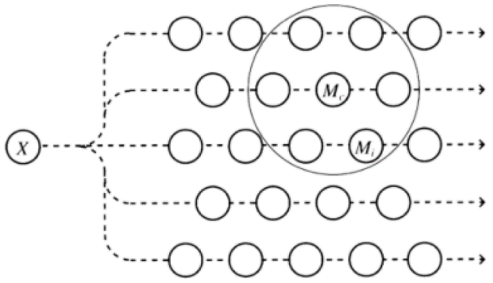


Fig. 3. A Self-Organizing Mapping Model.

Table 6
The Comparison of Some Articles on Task Scheduling.

Ref	Year	Ideas	Advantages	Disadvantages	software	Evaluation criteria
[39]	2010	Two-level scheduling	-Using more resources -Improving load balance	Not considering bandwidth and cost	Cloudsim	Makespan
[40]	2018	Using the containers	-Considering the real-time mode -Reducing task response time -Increasing the number of concurrent tasks in the fog node	- Not considering the execution time of tasks and network traffic -Removing cloud computing time - Removing the appropriate criterion for cloud	Libvirt Docker QEMU	Number of task concurrency Average reducing time
[41, 42]	2018	Using a Genetic Algorithm	-Considering the following cases -Dependencies of tasks -Data transmission delay -Response deadline	Not comparing with other methods	C++	Makespan Energy consumption
[43]	2014	Applying awareness-based competition method	-Performing well ineffective costs -Improving efficiency	Not performing well in some criteria	Java	Consumption costs Response time
[44]	2018	Using linear programming	Fog visual computing	Not considering the computational complexity	Matlab	Execution time
[45]	2020	Providing the algorithm that reduces costs under the Bag of Tasks job	- Consumption costs reduction -Reduction of task execution time -Applying all available resources	Not mentioned in the relevant article	Not stated	Consumption costs Execution time
[46]	2020	Clustering based on communication overhead	Increasing speed up values Decreasing schedule length ratio (SLR) Increasing performance	Clustering based on a criterion	Matlab	Speed up SLR
[47]	2018	Using heuristic methods (CPSO and FIPS) to schedule tasks	-Quick convergence -High precision -Easy implementation - Improve the QoS parameter -Better search capability	Not mentioned in the relevant article	Not stated	Throughput Delay Utilization
[48]	2019	Using the CPSO method along with scheduling algorithms to reduce the makespan	-Quick convergence -High precision -Easy implementation -Better search capability	Not mentioned in the relevant article	Matlab	Throughput Makespan Utilization Flowtime

input stimulates the elements, the element whose parameters most closely match this input is most stimulated. This element is called a "winner". Hence, the M_c (winning element) which is the most closely resembling x, makes all adjacent models to M_c, to modify their similarity to x [51].

As the neighboring models of the neuron or the winning element begin to resemble the x message, they try to resemble each other more; this means that all the models in the neighboring M_c become Smooth [52]. Different messages at different times affect different parts of the models set, whereby the M_i models after many training stages, have taken appropriate values, as with the initial x message, in the "signal space". These three steps, including the input propagation, the winning neurons selection, and the models' adaptation to the winning neurons' neighborhood, constitute the basic principles of a process. The latter stage is somehow non-parametric regression.

One of the most interesting features of self-organizing networks is that no supervision is required during the training time. In these networks, the output data is compared to what should be created based on an algorithm, and if the desired result is not achieved, the weight of the nodes will be changed to reach the final goal.

In short, the self-organizing neural network training algorithm is as follows:

1. The distance between the x pattern and all neurons.

$$d_{ij} = \left|\left|x_k - w_{ij}\right|\right| \tag{1}$$

2. Selecting the nearest neuron as the winning neuron

$$w_{ij} : d_{ij} = \min(d_{mn}) \tag{2}$$

3. Updating each neuron according to the neighborhood function

$$w_{ij} = w_{ij} + \alpha_h\left(w_{winner}, w_{ij}\right) \parallel x_k - w_{ij} \parallel \tag{3}$$

The α coefficient value reduces the impact of dissimilar weights.

1. This process is repeated until a specific stopping criterion is reached. Often the stopping criterion is a certain number of repetitions. In order to stabilize the convergence and stability of the map, the learning rate of the neighborhood radius in each iteration is reduced, whereby the convergence will tend to zero. The distance between the vectors is Euclidean distance, but other distances such as Mahalanobis and Manhattan can also be used [52].

3.2. Autoencoder

Autoencoder is a particular type of artificial neural network. Instead of network training and predicting the target Y's valueor input X, an Autoencoder is trained to reconstruct the input X [53]. Thus, the output vectors will have the same dimensions as the input vector. The general process of an Autoencoder is shown in Fig. 4. The Autoencoder is optimized by minimizing the reconstruction error where the corresponding code is the same learned feature.

In general, a single layer is not able to receive distinct features from raw data. Nowadays, researchers use deep Autoencoder, which sends the code learned from the previous Autoencoder to the next one to perform the task. Deep Autoencoder was developed by Hinton et al. [49] and is still widely studied in recent articles [50].

The Autoencoder has two sections, encoder, and decoder. In the encoder section, the input data is mapped to the latent space, and then the decoder section creates the input data. The latent space has a lower dimension than the input data. The Autoencoder can be used to reduce

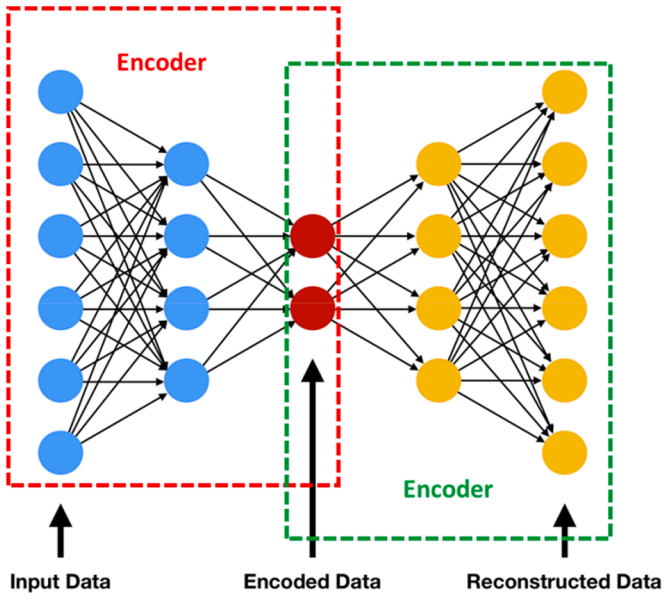


Fig. 4. The Autoencoder structure.

the feature. Given that the low-dimensional latent variables can reconstruct the input data, they will have the most important input properties. Thus, the Autoencoder can be used to extract the appropriate feature and to reduce data dimensions. There are other algorithms for extracting features such as Principal Component Analysis (PCA), Independent Component Analysis (ICA), Linear Discriminant Analysis (LDA) [54]. The difference between Autoencoder and the mentioned algorithms is that Autoencoder uses nonlinear conversion to map data dimensions from high to low. Further, the input data are received and compressed in the encoder section, while possible noises and useless information are deleted.

4. Two-phase scheduling (The proposed method)

The architecture proposed in this article has three layers of IoT, Fog and Cloud, as shown in Fig. 5. In the two-phase scheduling method, the clustering and scheduling sections are located in the fog layer. In this layer, the tasks' required information is received from the IoT layer and introduced into the clustering section. The task features are first introduced into the Autoencoder. New features are extracted; then, the SOM clusters them into two categories based on their execution location in the fog and cloud. After clustering, the tasks are sent to the scheduling section and are scheduled according to their execution location using Earliest-Deadline-First scheduling. The intended data tasks are sent to the computing section in the fog layer or the cloud layer. A section for the required data in the fog layer is considered in the architecture; whenever a task needs data to execute its commands, it could receive the data from the data section inside the fog.

To cluster and schedule tasks, the steps described in Fig. 6 are followed. These steps are described in detail in this section.

4.1. Getting the primary parameters of tasks

The Cloud-Fog system is responsible for executing all user requests within the deadline. The IoT devices send user requests to the fog for decision-making. Each request consists of a set of independent tasks. In this article, we have selected the effective scheduling parameters to construct the task format due to the study of articles in task scheduling. In this format, in addition to the features listed in Table 7, the device identification number is also considered. Each task is identified using the following multiples.

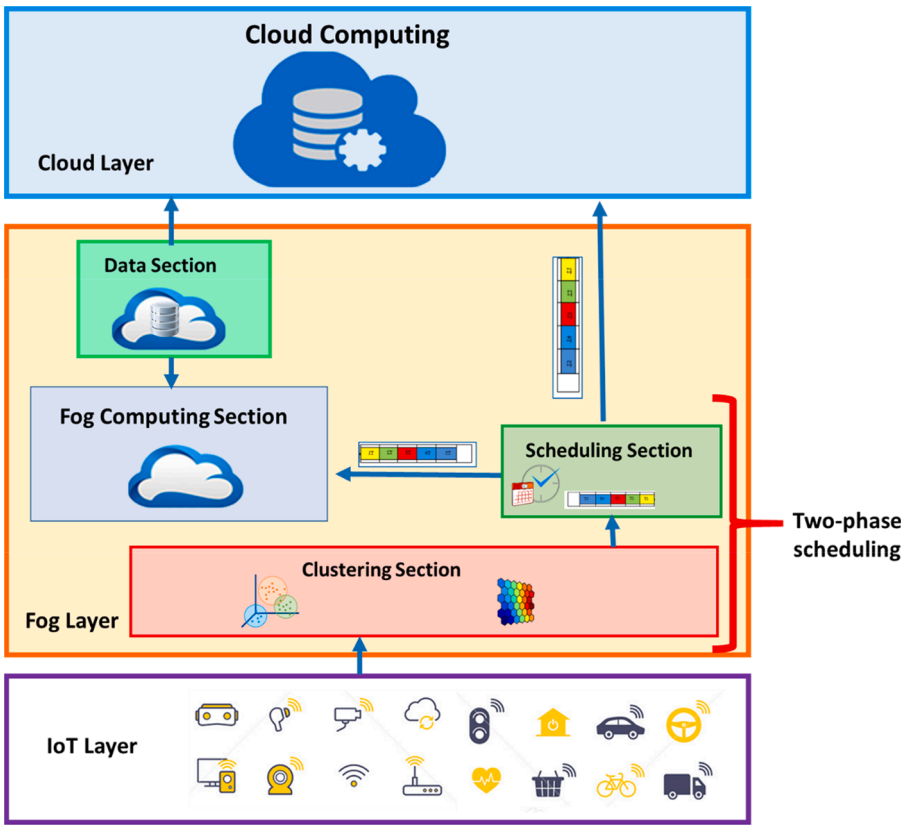


Fig. 5. The Structure of IoT with Cloud and Fog.

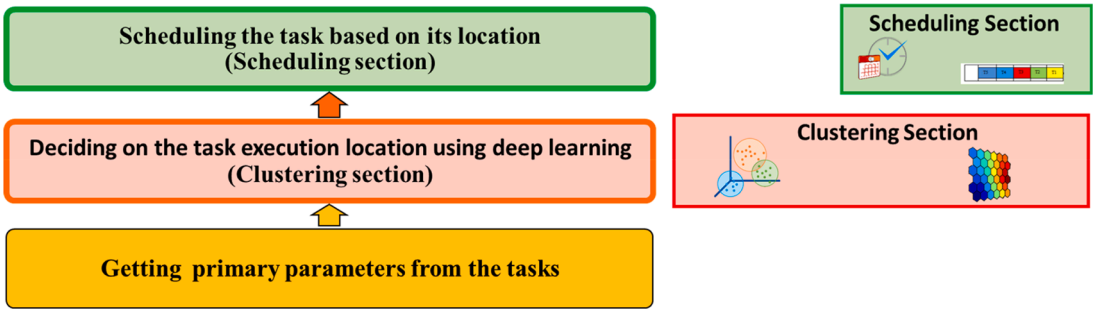


Fig. 6. Steps of the Proposed Method.

$$T < ID_{t_i}, T\_os_{t_i}, T\_ty_{t_i}, T\_pr_{t_i}, T\_priv_{t_i}, T\_hetr\_data_{t_i}, T\_deadline_{t_i}, T\_execution_{t_i}, T\_resource_{t_i}, T\_arvl_{t_i}, T\_data_{t_i}, T\_need\_data\ 1, \ldots, T\_need\_data\ 10 >$$

A database based on the simulation of real data is constructed to implement the proposed method. The method of constructing each of these features in the database is as follows. Different tasks are created in this data set, some of which require more processing, while others require more memory or bandwidth. Note that no datasets have been published in this research's literature section; therefore, we create this dataset.

- **The operating system type (T_os)**: it is the operating system from which the task originates. Four types of operating systems are considered in this study.
- **Task type (T_ty):** The task type is dependent on the scenario considered in IoT. For example, in the smart home scenario, the surveillance camera can have the motion detection task, other scenarios, storage and processing tasks, etc. Thus, first of all, the type of task should be specified to make the right decision to execute the task based on its type. In this study, according to the preliminary research, the task type is considered in three modes of memory, computing, and content distribution, distributed uniformly over 1,2,3 numbers in the task database.
- **Task priority (T_pr)**: The type of client can also influence the schedule decision-making. For example, in the smart health scenario, if a person with a particular disease changes, the schedule decision-making can be affected. Thus, sorting the clients and their tasks can determine the type and importance of that task. In the built database, the priority of tasks is between 1 and 5, and the higher the priority number, the more important the task.
- **Task privacy(T_priv)**: Consideration of this criterion and other criteria can help secure the data with the task. If the data security is high and can be processed in the fog, sending it to the cloud is unnecessary; however, if the fog cannot process the data, it should be encoded and sent to the cloud. The considered privacy level is between 1 and 4; level 1 has the highest privacy, while level 4 has the

Table 7
The dataset features.

Feature name	Specifications	Value range	The distribution type
Task_Os_Type	Operating system type	4 operating systems {1,2,3,4} Windows, Obent، ios، android	Uniform
Task_Type	Task type	{1,2,3} Memory,Computing Content distribution	Uniform
Task_privacy	Level of Confidentiality	4 types (1) The highest level of confidentiality (4) The lowest level of confidentiality	Uniform
Task_deadline	Execution deadline	[1 20]ms	Uniform
Task data heterogeneity	Data heterogeneity type	{1,2,3,4,5} Video data, audio data, text data, multimedia data and sensor information	Uniform
Task Execuation	Task execution time	[1 60] ms	Uniform
Task Resource	Required resources	{1,…,8}	Uniform
Task arrival	Task arrival time		Poisson with an average interval of 5 milliseconds
Task data	Task required data	{1 … 10000}bit	Zipf
Task required data	The maximum number of required data	{1 … 10}	—

lowest rate. The level of privacy with a uniform distribution between 1 and 4 has been created in this database.

- **Task data heterogeneity (T_hetr_data):** This section specifies the extent of heterogeneity of the data task using a uniform distribution within a range of 1 to 5. Numbers 1 to 5 represent the type of image data, audio data, text data, multimedia data, and sensor information, respectively.
- **Task deadline (T_deadline):** It indicates the maximum acceptable time to complete a task. Performing the task after its deadline will be valueless or undervalued. An appropriate scheduler algorithm should schedule and execute the maximum number of possible tasks according to their priority over the expected time intervals. Thus, each task should have a deadline, whereby the scheduler can make the right decision to execute it on the cloud or the fog. The deadline is between 1 and 20 milliseconds, with a uniform distribution.
- **Task execution (T_execution):** The execution time is between 1 and 60 milliseconds, distributed uniformly.
- **Task Resources (T_resource):** It specifies the required computational and storage resources. The number of required resources in the built database is distributed uniformly between 1 and 8.
- **Task arrival time (T_arvl):** Task arrival time for the scheduler is also determined by the Poisson distribution with an average arrival time of five milliseconds.
- **Task Data and maximum required task data:** The first 10,000 data with a size of 200 to 500,000 bits are generated using uniform distribution to create the necessary data associated with each task. Then, we used the Zipf distribution to assign each data to each task. The maximum number of required data is generated by the Zipf distribution, and after removing duplicate data numbers, that data is considered for the intended task.

4.2. Deciding on the task execution location (clustering section)

In various IoT scenarios, huge amounts of tasks with various features and data from IoT devices have been created, where these tasks and data should be collected and properly managed. To manage the tasks better, before scheduling, they can be clustered using some techniques according to the system's criteria. In this article, the Self-Organizing Map clustering method is used before the scheduling section. Self-organizing mapping (SOM) is an unsupervised clustering method that transforms multidimensional and complex data spaces into a two-dimensional space. In this article, this method has been used in three different ways for task clustering. The following SOM-based methods have been studied to achieve the appropriate clustering method.

Method 1: Task clustering by self-organizing map
Method 2: Task clustering by hierarchical self-organizing map
Method 3: Task clustering by Autoencoder and self-organizing map

The details of the implementation steps of the above methods are described in this section.

Method 1: task clustering by self-organizing map (SOM)

In this method, the parameters obtained from the tasks are sent directly to the SOM for clustering. For SOM training, 2×1 hexagonal topology is used. The distance criterion for the neurons' neighborhood is the Euclidean distance. The maximum epoch for SOM training is 1000 times. Fig. 7a reveals this topology where each hexagon represents a neuron. The SOM input vector has ten dimensions, including the task parameters described in the previous section. Fig. 7b displays the network architecture, including the input matrix, the output mapping, the layer containing neurons, and the neurons' weight vector(W).

Method 2: task clustering by hierarchical self-organizing map (H-SOM)

The architecture used in the hierarchical self-organizing map is displayed in Fig. 8. In this architecture, a Self-Organizing Map is used in every layer, and the output of this network enters the next layer as a new feature, along with the previous data. Self-Organizing Map with Hexagonal grid with dimensions of 4×4, $3 \times 2 \times 2$, 3×2, and 2×2 and 2×1 are considered in this architecture, respectively.

Method 3: task clustering by autoencoder and self-organizing map (AE-SOM)

In this section, before clustering the task parameters, the features are extracted, resulting in feature reduction. The Autoencoder uses three or more layers; the input layer receives the task parameters, some hidden layers, and the output layer in which each neuron has the same meaning as the input one. The Autoencoder consists of two sections, encoder, and decoder. In this method, the encoder section of the Autoencoder is used to extract the features. The three-layer, two-layer, and single-layer modes are examined for the hidden layer of the Autoencoder. Fig. 9 displays one of the structures used in this method. In this structure, after receiving features from the task, the data is sent to the first layer, which is the encoder section of the Autoencoder. The Autoencoder has 25 neurons and is trained with Logistic sigmoid transfer function for the encoder section and a Positive saturating linear transfer function for the decoder one. The second layer of the encoder section is another Autoencoder with 15 neurons. It is trained with a Positive saturating linear transfer function for the encoder section plus a Logistic sigmoid transfer function for the decoder one. In the third layer, the encoder section is an Autoencoder with 5 neurons, where in both the encoder and decoder sections, the Logistic sigmoid transfer function is considered. After extracting the features, the SOM with two-in-one hexagonal topology clusters them. The distance criterion for the neurons

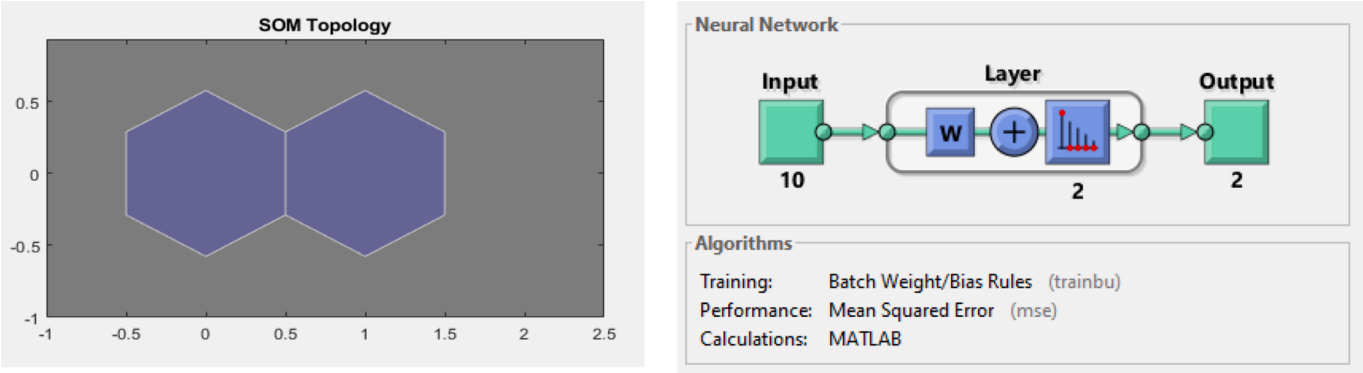


Fig. 7. (a) SOM Topology, (b) SOM Architecture.

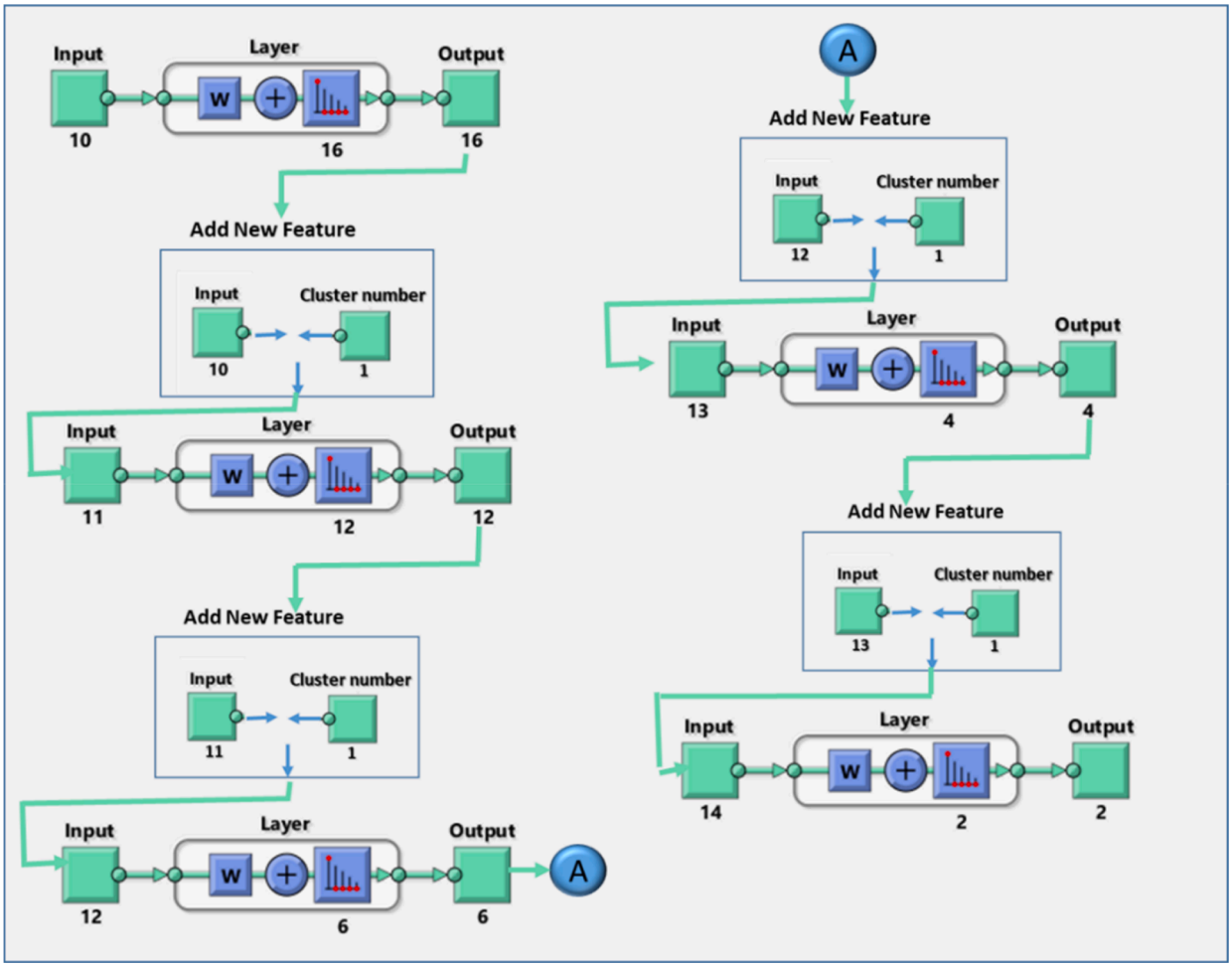


Fig. 8. Hierarchical Self-Organizing Map.

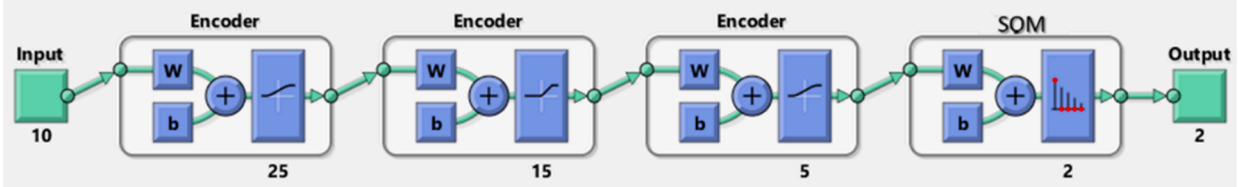


Fig. 9. The Architecture of Autoencoder with 3 Layers and the SOM Layer.

neighborhood in SOM is the Euclidean distance. The Maximum epoch for SOM training is 1000 times. The SOM clusters the input data into two categories.

In the proposed method, we have examined different modes for the Autoencoder structure. Table 8 outlines the number of layers and the number of neurons in each layer. The number of neurons in layers is examined via the trial and error method. In the structures with 10 neurons in the last layer, the Autoencoder is used to produce new features. The feature reduction is not discussed, and to reduce the features, the last layers are considered 5 and 3.

4.3. Earliest-deadline-first scheduling (Scheduling section)

The tasks in each cluster are scheduled by the Earliest-Deadline-First (EDF) algorithm and sent to the cloud. The EDF algorithm is a dynamic priority schedule used in real-time operating systems. Whenever scheduling is required, this algorithm searches the tasks, selects, and runs the one in the queue whose deadline is near to end [55]. The reason for using the EDF algorithm is that it is optimal. The optimization of the EDF algorithm among all scheduling algorithms has been proven on a uni-processor. It means that, if EDF does not schedule a set of tasks in real-time, this task set cannot be scheduled by any algorithm [56]. Some of the disadvantages of this algorithm are its low predictability, low control over performance, and high overhead.

5. Simulation results

The features of the simulated environment are reported in Table 9. The Python language has also been used to simulate the proposed method.

The data set created and used in this research has 100,000 independent tasks and 10,000 data. The maximum number of data associated with each task is 10, each of which has a different size on a bit basis. The task arrival time to the scheduler is set by the Poisson distribution with an average arrival distance of five milliseconds. Data transfer latency is also considered zero. Table 10 indicates the dataset characteristics.

The results of the simulation of the proposed method are presented in two sections.

5.1. The result of the clustering section

The significant feature of learning methods is that they need appropriate data to train the model and obtain the required parameters. In this section, we have used 70,000 random data for the training section and 30,000 data for the test section to train the clustering methods mentioned in this article. This division of training and testing is used only in the clustering section, while for scheduling, the results are stated for all research tasks.

As shown in Section 3, three clustering methods given based on the SOM are presented in this article. The proposed system should be analyzed after scheduling to choose the appropriate clustering method. In the method number 3, mentioned in the clustering section, the

Table 8
The Number of Layers and Neurons in Each Layer.

Number of Autoencoder layers	Number of neurons in each layer
3	25-15-5
3	50-25-10
2	40-10
2	50-10
2	56-10
2	8-5
1	10
1	5
1	3

Table 9
Simulation Setup.

System	Intel R Core*TM* i5-6200U, CPU 2.30GHz
Memory	4 GB
Simulator	Matlab 2019
Operating system	Windows 10 Professional

Table 10
Simulation and Dataset Characteristics.

Parameter	
Number of Cloud	1
Cloud CPU rate	10,000 MIPS
Number of Fog	1
Fog CPU rate	1000 MIPS
Number of Tasks	100,000
Number of Data	10,000
Maximum of the number of data required for each task	10
Minimum data on the bit	200 bit
Maximum data size on the bit	5000 bit
Task arrival time	5ms
Number of the feature of Task	10
Features of Task	• The operating system type • Task type • Task priority • Task privacy • Task data heterogeneity • Task deadline • Task execution • Task Resources • Task arrival time • Task Data and Maximum required task data

Autoencoder is applied to extract features. Different cases are examined to research and study the number of layers required to construct the Autoencoder and determine which architecture is more appropriate. For further study, we have also considered the cases in which the Autoencoder has increased the dimension to make sure whether the use of the Autoencoder will be useful or not. Whenever the information about the ideal clustering is not available for the data set, we should use an inherent method to evaluate the clustering quality. In general, intrinsic methods investigate and evaluate a clustering method by examining the separation of clusters and their inside compression. Many intrinsic methods use a similarity measurement between objects in the dataset. In order to measure the quality of clustering, we have used the average value of Silhouette Coefficient (SC) [57] of all objects in the datasets. The SC method, one of the most common and useful validation methods for the clustering algorithm, was first proposed by Kaufman & Rousseeuw in 1990 [58]. This coefficient is calculated by Eq. 4.

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}} \tag{4}$$

This coefficient is between -1 and 1. The value of a(x) reflects the cluster's compression in which the object x is located. A smaller amount for a(x) indicates more cluster compression. The value of b(x) denotes the degree of x separation from the other clusters. A larger value for b(x) indicates that x is farther away from the other clusters. Its scoring range is between 0 and +1; in a way that, if the measurement is in the range of +1, the desired cluster is very far from the adjacent cluster. On the other hand, the zero state indicates no difference between the intended cluster and the neighboring clusters. Finally, the -1 state indicates the intended cluster is incorrectly assigned.

Table 11 reports the different types of Autoencoder architecture by SC. The Autoencoder of a layer has a higher silhouette coefficient for training and testing data, as shown in the table. According to the results obtained from the silhouette coefficient, from now on, we have researched the Autoencoder models, which have a silhouette coefficient

Table 11
Comparison of different types of Autoencoder architecture by SC in AE-SOM method.

Number of Autoencoder layers	Number of neurons in each layer	Silhouette Coefficient – Training	Silhouette Coefficient - Test
3	25-15-5	0.5877	0.5896
3	50-25-10	0.5865	0.5880
2	40-10	0.5857	0.5788
2	50-10	0.5785	0.5812
2	56-10	0.5796	0.5820
2	8-5	0.5895	0.5910
1	10	0.5900	0.5914
1	5	0.5941	0.5957
1	3	0.5834	0.5851

above 0.58, and we have also selected Autoencoder models that reduce the features. The chosen methods are highlighted in Table 11.

We examine the scattering of the methods in the clusters. Table 12 summarizes the scattering rates in clusters with SOM, the H-SOM, and the AE-SOM method.

The distribution of tasks to the two clusters of fog and cloud in the AE-SOM method with different layers is almost the same as the SOM method, but the H-SOM method has less distribution in the fog section.

Table 13 compares the execution time of algorithms for clustering tasks. The Autoencoder method, with more hidden layers and more depth, takes more time to train. In deep autoencoders, each layer of the Autoencoder should first be trained, and then its encoder section should be used to create the next layer input. As such, the AE-SOM (5-15-25) training time is longer than other algorithms. The H-SOM method takes more time to train than the SOM method. Because it also trains multiple SOM layers hierarchically and uses the previous layers' data for the next one. Among the mentioned methods in Table 13, the SOM method has the least time for training and the least time for testing, and the AE-SOM (3) is in second place. Thus, the AE-SOM method requires a very short operating time and is close to real-time.

5.2. The results of the scheduling section

After the tasks are clustered, we use the EDF method to schedule them. To evaluate the proposed methods, we measured the execution time of fog and cloud, the missed rate of tasks, and the cost.

5.2.1. Examining the execution time of fog and cloud

After clustering and scheduling tasks, they are transferred to their execution location and executed in the intended location (fog or cloud). Table 14 reports the total time required to execute the tasks transferred to the cloud and fog in seconds. In this article, the executive power of cloud compared to fog is ten times, whereby it can perform more tasks per second. According to Table 14, the H_SOM method took 157.81 seconds to complete all tasks within the cloud cluster, which is the shortest time required to perform tasks in the cloud, among other methods. However, to perform tasks in the fog cluster, it took the longest time among other methods. According to Table 12, which shows the scattering of tasks among the clusters, the reason for the increase in the total execution time of tasks in the fog cluster is that the clustering method leads to more tasks in this cluster. Accordingly, all the methods have almost the same distribution except the H-SOM.

Table 12
The Rate of Scattering in Clusters.

Method	Cloud(%)	Fog(%)
SOM	49.80	50.20
H-SOM	35.62	64.38
AE-SOM (25-15-5)	49.12	50.88
AE-SOM (8-5)	53.25	46.75
AE-SOM (5)	49.86	50.14
AE-SOM (3)	46.42	53.58

Table 13
The Execution Time of Clustering Algorithms.

Method	Train Time (s)	Operating Time (Test Time)(s)
SOM	22.11	0.058
H-SOM	199.19	0.116
AE-SOM (25-15-5)	455.03	0.164
AE-SOM (8-5)	125.06	0.132
AE-SOM (5)	103.35	0.096
AE-SOM (3)	94.62	0.082

Table 14
The execution time of all tasks within the cloud and fog cluster.

Method	Cloud		Fog		Cloud & Fog
	Total Execution time (s)	Ave. Task Execution Time (s)	Total Execution time (s)	Ave. Task Execution Time (s)	Ave. Task Execution Time (s)
SOM	286.6254	0.0058	7647.039	0.1523	0.0765
H-SOM	157.8196	0.0044	8935.097	0.1388	0.0894
AE-SOM (25-15-5)	280.371	0.0057	7709.583	0.1515	0.0771
AE-SOM (8-5)	324.1522	0.0061	7271.771	0.1556	0.0727
AE-SOM (5)	287.2278	0.0058	7641.015	0.1524	0.0764
AE-SOM (3)	722.6291	0.0156	3287.002	0.0614	0.0329

Nevertheless, among these methods, the AE-SOM(25-15-5), in 280.371 seconds, performs all the tasks within its cloud, which is the shortest time among the other methods. The AE-SOM(3) performs all the cloud cluster tasks in 722.62 seconds, spending 3287 seconds to perform all its tasks in the fog cluster. It is the shortest time to complete all the tasks in the fog, among the other methods. The reason is that the tasks with high processing needs are in the cloud cluster, while those with low processing needs are in the fog cluster. In general, the AE-SOM(3) requires a shorter time to perform all tasks in both clusters. Accordingly, the total execution time and the average execution time of each task are reduced.

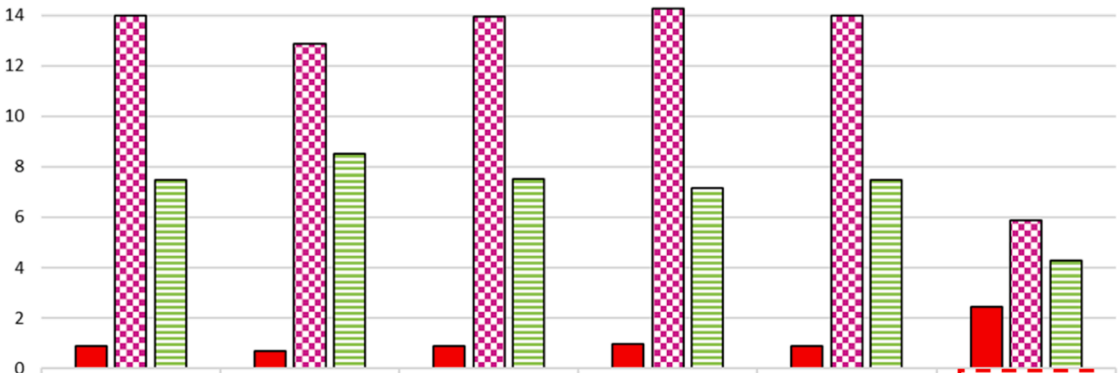
5.2.2. Examining the rate of missed tasks

Fig. 10 reveals the percentage of missed tasks. As shown in the table with the dotted line, the AE-SOM method with a layer containing three neurons has a lower missed rate than the SOM methods and the hierarchical SOM. The AE-SOM method has placed tasks in the fog cluster through feature extraction and reduction, 94.13% of which have been executed in the fog without any problems. Although 97.56% of tasks have been performed in the cloud cluster, in general, this clustering has been able to reduce the missed rate throughout the system. The AE-SOM method is 4.23% less than the highest missed rate, 2.89% less than the lowest missed rate, and 3.35% less than the average missed rate of the compared methods. According to the study's results, the use of feature extraction and its reduction can be considered successful in deciding on the location of task execution.

5.2.3. Examining the cost

Fog-cloud computing infrastructure includes processors such as cloud nodes and fog nodes. Each one has the same characteristics, such as CPU rate, CPU usage cost, memory usage cost, and bandwidth usage cost. Cloud nodes are more powerful than fog nodes, though they incur more costs.

In this article, it is assumed that the connection between sensors and the device is not considered. As the communications between fog nodes



	SOM	H-SOM	AE-SOM(25-15-5)	AE-SOM(8-5)	AE-SOM(5)	AE-SOM(3)
Cloud_missed_rate (%)	0.9	0.67	0.89	0.96	0.9	2.44
Fog_missed_rate (%)	13.99	12.85	13.93	14.26	14	5.87
Cloud_Fog_missed_rate (%)	7.473	8.511	7.525	7.175	7.468	4.277

Fig. 10. The Percentage of the Missed Task.

occur in a small or medium-sized urban area, they are established according to wireless grid networks' standards. Thus, we have considered a small cost for them. However, communications between the fog nodes and cloud servers have a delay and high costs. Fig. 11 reveals communication costs.

In the proposed method, the costs are calculated in Grid dollars (G $) - the simulation's currency to replace the real money. Table 15 indicates the considered costs.

The cost of this study is calculated by Eq. 5.

$$Total\ Cost = \sum_{T_i \varepsilon T} Cost(T_i) \tag{5}$$

The total processing and memory usage costs and the bandwidth for each task are calculated by Eq. 6.

$$Cost(T_i) = C_p(T_i) + C_m(T_i) + C_b(T_i) \tag{6}$$

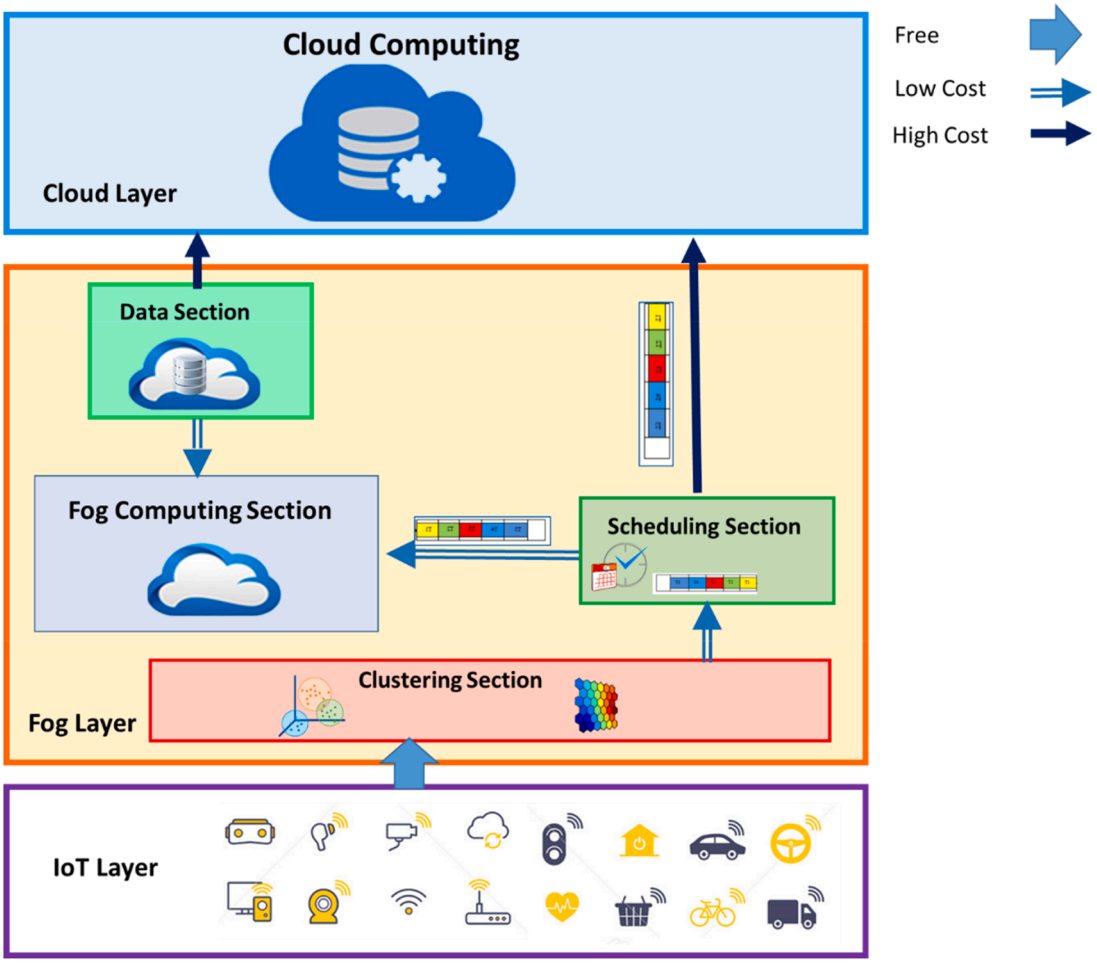


Fig. 11. The Communication Costs in the Proposed System.

Table 15
The cost details.

Parameter	Fog	Cloud	Unit
CPU rate	1000	10000	**MIPS**
CPU usage cost	0.1	0.7	**G$/s**
Memory usage cost	0.01	0.02	**G$/MB**
Bandwidth usage cost	0.001	0.01	**G$/MB**

The detail of calculating the cost is indicated in the following formulas.

1. Processing costs for each task

$$C_p(T_i) = c_1 * ExeTime(T_i) \tag{7}$$

c_1 : The cost of using the CPU per time unit is in the cloud.

2. The cost of using memory for each task

$$C_m(T_i) = c_2 * Mem(T_i) \tag{8}$$

c_2 : The cost of using memory for each data unit

3. The cost of using bandwidth for each task

$$C_b(T_i) = c_3 * BW(T_i) \tag{9}$$

Table 16 presents the total costs in the cloud, fog, and both of them. As shown in Table 16, in the AE-SOM method, the total cloud and fog cost is 347.1 (G$) less than via the H-SOM method and 264.4 (G$) less than through the SOM method.

6. Conclusion

To improve the response performance of tasks on the IoT, they were first clustered by the three SOM based clustering methods. Then, the EDF algorithm was used to schedule tasks in the cloud or fog. In the first method, the SOM divided the primary features of tasks into two clusters. In the second one, the H-SOM, the primary features of tasks were given to the hierarchical SOM cluster. In the third method, the AE-SOM, we used a single-layer Autoencoder with three neurons. In this method, the features were extracted from the tasks' initial features and then given to the SOM for clustering. Considering the simulation of the three mentioned methods, we concluded that the SOM and H-SOM methods are not appropriate in terms of missed task rate and cost, while the AE-SOM method has a better result. This method reduced the number of missed tasks by 3.19% compared to the SOM method and by 4.23% compared to the H-SOM method. Further, costs, including processing, memory, and bandwidth costs, were 305.75 (G$) less than the two compared methods' average cost. In the proposed method, data transfer latency was not considered, and the tasks were independent. For future work, the proposed method can be applied to dependent tasks by considering the transfer delay.

Further, by increasing the number of fog nodes and establishing communication between them, the number of clusters in the decision-making section can be changed to multiple clusters. The proposed method can be examined. The decision-making and scheduling sections can be applied in a distributed manner in all fogs to solve the bottleneck problem. The device sends the task to its nearest fog, with that fog performing the task inside itself or another fog, whereby the survivability services can also be insured.

Table 16
The Total Costs.

Method	Cloud_cost (G$)	Fog_cost (G$)	Cloud_Fog_cost (G$)
SOM	291.18	958.43	1249.61
H-SOM	176.68	1155.63	1332.31
AE-SOM(3)	539.22	445.99	985.21

CRediT authorship contribution statement

Shabnam Shadroo: Conceptualization, Validation, Investigation, Writing - original draft, Writing - review & editing, Visualization. **Amir Masoud Rahmani:** Conceptualization, Validation, Investigation, Writing - review & editing, Visualization, Supervision, Project administration. **Ali Rezaee:** Validation, Writing - review & editing.

Declaration of Competing Interest

The authors declare that there is no conflict of interest regarding the publication of this article

References

[1] J. Bradley, J. Barbier, D. Handler, Embracing the Internet of Everything, Cisco, USA, 2013.
[2] L. Atzori, A. Lera, G. Morabito, The Internet of Things: a survey, Comput. Netw. 54 (15) (2010) 2787–2805.
[3] Sh. Shadroo, A.M. Rahmani, Systematic survey of big data and data mining in internet of things, Comput. Netw. 139 (2018) 19–47.
[4] A. Gubbi, T. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): a vision, architectural elements, and future directions, Fut. Gener. Comput. Syst. 29 (2013) 1645–1660.
[5] A. Malik, H. Om, Cloud computing and Internet of Things integration: architecture, applications, issues, and challenges, in: W. Rivera (Ed.), Sustainable Cloud and Energy Services, Springer, Cham, 2018, pp. 1–24.
[6] R. Mahmud, R. Kotagiri, R. Buyya, Fog computing: a taxonomy, survey and future directions. Internet of Everything. Internet of Things (Technology, Communications and Computing), Springer, Singapore, 2017, pp. 103–130.
[7] I. Ullah, H.Y. Youn, Task classification and scheduling based on K-means clustering for edge computing, Wirel. Pers. Commun. (2020).
[8] A. Shahidinejad, M. Ghobaei-Arani, M. Masdari, Resource provisioning using workload clustering in cloud computing environment: a hybrid approach, Cluster Comput. (2020).
[9] A. Asensio, X. Masip-Bruin, R.J. Durán, et al., Designing an efficient clustering strategy for combined Fog-to-Cloud scenarios, Fut. Gener. Comput. Syst. 109 (2020) 392–406.
[10] H. Shah-Mansouri, V.W.S. Wong, Hierarchical fog-cloud computing for IoT systems: a computation offloading game, IEEE Internet Things 5 (4) (2018) 3246–3257.
[11] P. Vinothiyalakshmi, R. Anitha, Workload mining in cloud computing using extended cloud Dempster–Shafer theory (ECDST), Wirel. Pers. Commun. (2020).
[12] p. Hosseinioun, M. Kheirabadi, S.R. Kamel Tabbakh, R. Ghaemi, A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm, J. Parallel Distrib. Comput. (2020).
[13] C.D. Gómez Romero, J.K. Díaz Barriga, J.I. Rodríguez Molano, Big Data Meaning in the Architecture of IoT for Smart Cities, Springer International Publishing, 2016.
[14] p. Zhang, M. Zhou, G. Fortino, Security and trust issues in Fog computing: a survey, Fut. Gener. Comput. Syst. 88 (2018) 16–27.
[15] M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, Deep learning for IoT big data and streaming analytics: a survey, IEEE Commun. Surv. Tutor. 20 (4) (2018) 2923–2960.
[16] J. Wang, Y. Ma, L. Zhang, R.X. Gao, D. Wu, Deep learning for smart manufacturing: methods and applications, J. Manuf. Syst. 48 (2018) 144–156.
[17] Y. Guo, Y. Liu, et al., Deep learning for visual understanding: a review, Neurocomputing 187 (2016) 27–48.
[18] R.K. Behera, S. Gupta, A. Gautam, Big-data empowered cloud centric Internet of Things. International Conference on Man and Machine Interfacing (MAMI), 2015.
[19] S. Sowe, T. Kimata, M. Dong, K. Zettsu, Managing heterogeneous sensor data on a big data platform: IoT services for data-intensive science, in: IEEE 38th Annual International Computers, Software and Applications Conference, Sweden, 2014.
[20] A. Paul, A. Ahmad, M.M. Rathore, S. Jabbar, Smartbuddy: defining human behaviors using big data analytics in social internet of things, IEEE Wirel. Commun. (2016).
[21] D.G. Páez, F. Aparicio, M. de Buenaga, J.R. Ascanio, Big data and IoT for chronic patients monitoring, in: International Conference on Ubiquitous Computing and Ambient Intelligence, 2014.
[22] Y. Ma, Y. Wang, Yang, et al., Big health application system based on health internet of things and big data, IEEE Access 5 (2017) 7885–7897.

[23] G. Suciu, A. Vulpe, O. Fratu, V. Suciu, M2M remote telemetry and cloud IoT big data processing in viticulture, in: International Wireless Communications and Mobile Computing Conference (IWCMC), 2015.
[24] G. Suciu, V. Suciu, S. Halunga, O. Fratu, Big data, internet of things and cloud convergence for E-health applications," *new contributions in information systems and technologies*. Adv. Intell. Syst. Comput. 353 (2015).
[25] G. Suciu, V. Suciu, A. Martian, et al., Big data, internet of things and cloud convergence – an architecture for secure E-health applications, J. Med. Syst. (2015).
[26] M. Vögler, J.M. Schleicher, C. Inzinger, S. Dustdar, Ahab: a cloud-based distributed big data analytics framework for the Internet of Things, Software (2017).
[27] M.T. Khorshed, N.A. Sharma, K. Kumar, M. Prasad, Integrating Internet-of-Things with the power of cloud computing and the intelligence of big data analytics : a three layered approach, in: Computer Science and Engineering (APWC on CSE), 2015 2nd Asia-Pacific World Congress on, Fiji, 2015.
[28] M. Mozumdar, A. Shahbazian, N.Q. Ton, A big data correlation orchestrator for Internet of Things. IEEE World Forum on Internet of Things (WF-IoT), 2014.
[29] P. Dineshkumar, R. SenthilKumar, K. Sujatha, R.S. Ponmagal, V.N. Rajavarman, Big data analytics of IoT based health care monitoring system, in: *IEEE Uttar Pradesh Section International Conference on Electrical,Computer and Electronics Engineerin*, 2016.
[30] G. Manogaran, D. Lopez, C. Thota, K.M. Abbas, S. Pyne, R. Sundarasekar, Big data analytics in healthcare internet of things. Innovative Healthcare Systems for the 21st Century, Springer, 2017, pp. 263–284.
[31] I. Kholod, M. Kuprianov, I. Petukhov, Distributed data mining based on actors for Internet of Things, in: 2016 5th Mediterranean Conference on Embedded Computing (MECO), 2016.
[32] H. Li, K. Ota, M. Dong, Learning IoT in edge: deep learning for the internet of things with edge computing, IEEE Netw. 32 (1) (2018).
[33] P. Li, Z. Chen, L.T. Yang, et al., Deep convolutional computation model for feature learning on big data in internet of things, IEEE Trans. Ind. Inf. 14 (2) (2018) 790–798.
[34] N. Elaraby, M. Elmogy, S.H. Barakat, Large scale sensor data processing based on deep stacked Autoencoder network, J. Theor. Appl. Inf. Technol. 95 (21) (2017).
[35] M. Mohammadi, A. Al-Fuqaha, et al., Semisupervised deep reinforcement learning in support of IoT and smart city services, IEEE Internet Things J. 5 (2) (2018) 624–635.
[36] Y. Lv, Y. Duan, W. Kang, Z. Li, F. Wang, Traffic flow prediction with big data: a deep learning approach, IEEE Trans. Intell. Transp. Syst. 16 (2) (2015) 865–873.
[37] D.C. Mocanu, E. Mocanu, P.H. Nguyen, M. Gibescu, A. Liotta, Big IoT data mining for real-time energy disaggregation in buildings, in: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2016.
[38] F. Alam, R. Mehmood, I. Katib, A. Albeshri, Analysis of eight data mining algorithms for smarter internet of things (IoT). Procedia Computer Science, 2016.
[39] Fang, Y., Wang, F., Ge, J., "A Task Scheduling Algorithm Based on Load Balancing in Cloud Computing," In: Wang F.L., Gong Z., Luo X., Lei J. (Eds.) Web Information Systems and Mining, Lecture Notes in Computer Science, vol. 6318. Springer, Berlin, Heidelberg, 2010.
[40] L. Yin, J. Luo, H. Luo, Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing, IEEE Trans. Ind. Inf. 14 (10) (2018) 4712–4721.
[41] C. Tang, M. Hao, X. Wei, Energy-aware task scheduling in mobile cloud computing, Distrib. Parallel Databases 36 (3) (2018) 529–553.
[42] C. Tang, S. Xiao, X. Wei, M. Hao, W. Chen, Energy-efficient and deadline-satisfied task scheduling in mobile cloud computing, in: IEEE International Conference on Big Data and Smart Computing, Shanghai, 2018.
[43] Hung, P.P., Bui, TA., Huh, EN., "A New Approach for Task Scheduling Optimization in Mobile Cloud Computing," In: Park J., Zomaya A., Jeong HY., Obaidat M. (eds) Frontier and Innovation in Future Computing and Communications, Dordrecht, Lecture Notes in Electrical Engineering, vol 301. Springer, 2014.
[44] H.M. Chu, S.W. Yang, P. Pillai, Y.K. Chen, Scheduling in visual fog computing: NP-completeness and practical efficient solutions, in: AAAI Publications, Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[45] D. Tychalas, K. Helen, A scheduling algorithm for a fog computing system with bag-of-tasks jobs: simulation and performance evaluation, Simul. Modell. Pract. Theory 98 (101982) (2020).
[46] Q. Tian, J. Li, D. Xue, A hybrid task scheduling algorithm based on task clustering, Mobile Netw. Appl. (2020).
[47] F. Al-Turjman, M.Z. Hasan, H. Al-Rizzo, Task scheduling in cloud-based survivability applications using swarm optimization in IoT, Trans. Emerging Tel. Tech. (2018) e3539.
[48] MZ Hasan, H. Al-Rizzo, Task scheduling in Internet of Things cloud environment using a robust particle swarm optimization, Concurrency Computat. Pract. Exper. (2019) e5442.
[49] T. Kohonen, Self-Organizing Maps, Springer-Verlag, Berlin Heidelberg, 2001.
[50] L. Li, Y. Wang, What drives the aerosol distribution in Guangdong-the most developed province in Southern China? Sci. Rep. 4 (2014).
[51] J. Vesanto, SOM-based data visualization methods, Intell. Data Anal. 3 (2) (1999) 111–126.
[52] T. Kohonen, S. Kaski, H. Lappalainen, Self-organized formation of various invariant feature filters in the adaptive-subspace SOM, Neural Comput. 9 (1997) 1321–1344.
[53] C.Y. Liou, W.C. Cheng, J.W. Liou, D.R. Liou, Autoencoder for words. Neurocomputing, Neurocomputing 139 (2014) 84–96.
[54] V. Neagoe, A. Mugioiu, I. Stanculescu, Face Recognition using PCA versus ICA versus LDA cascaded with the neural classifier of concurrent self-organizing maps, in: 2010 8th International Conference on Communications, Bucharest, 2010.
[55] G. Gupta, V.K. Kumawat, P.R. Laxmi, D. Singh, et al., A simulation of priority based earliest deadline first scheduling for cloud computing system, in: 2014 First International Conference on Networks & Soft Computing (ICNSC2014), Guntur, 2014.
[56] S.K.F. Ali, M.B. Hamad, Implementation of an EDF algorithm in a cloud computing environment using the CloudSim tool, in: International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering, Khartoum, 2015.
[57] H.B. Zhou, J.T. Gao, Automatic method for determining cluster number based on silhouette coefficient, Adv. Mater. Res. 951 (2014) 227–230.
[58] L. Kaufman, P. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, 1990.

Shabnam Shadroo was born in Mashhad, Iran, in 1983. She received a B.S. degree in hardware computer engineering from the University of Sajad, Mashhad, Iran, in 2005, and the M.S degrees in artificial intelligence engineering from the Islamic Azad University, Mashhad, a student in software computer engineering in the Science and Iran, in 2007. She is Ph.D. Research Branch of the Islamic Azad University, Tehran, Iran, in 2016. She joined the Department of software computer Engineering, Azad University of Mashhad, as a Lecturer, in 2006 and was promoted to a Faculty member in 2009. Her current research interests include machine vision, Big data, and IoT.

Amir Masoud Rahmani received his BS in Computer Engineering from Amir Kabir University, Tehran, in 1996, the MS in Computer Engineering from Sharif University of Technology, Tehran, in 1998, and the PhD degree in Computer Engineering from IAU University, Tehran, in 2005. Currently, he is a Professor in the Department of Computer Engineering at the IAU University. He is the author/co-author of more than 300 publications in technical journals and conferences. His research interests are in the areas of distributed systems, the Internet of things, and evolutionary computing.

Ali Rezaee is an assistant professor with the Science and Research Branch of Islamic Azad University (IAU). He received his B.S., and M.S. degrees in software engineering in 2005, and 2008 respectively. In 2012, he received his Ph.D. degree in Software Systems from IAU. He is the chair of Distributed Systems laboratory, and Formal Verification laboratory at IAU. He is also lead system architect at Open Farm Technologies. His main research interests include distributed systems, IoT, formal verification, operating systems, and software architecture.